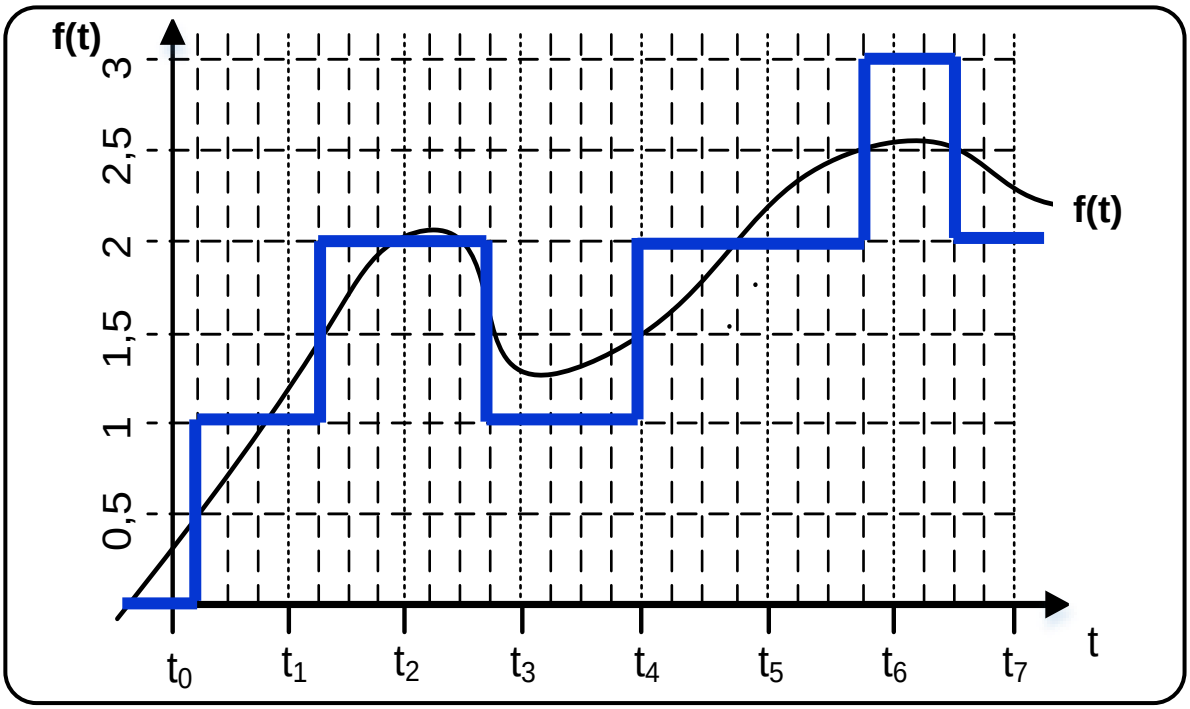
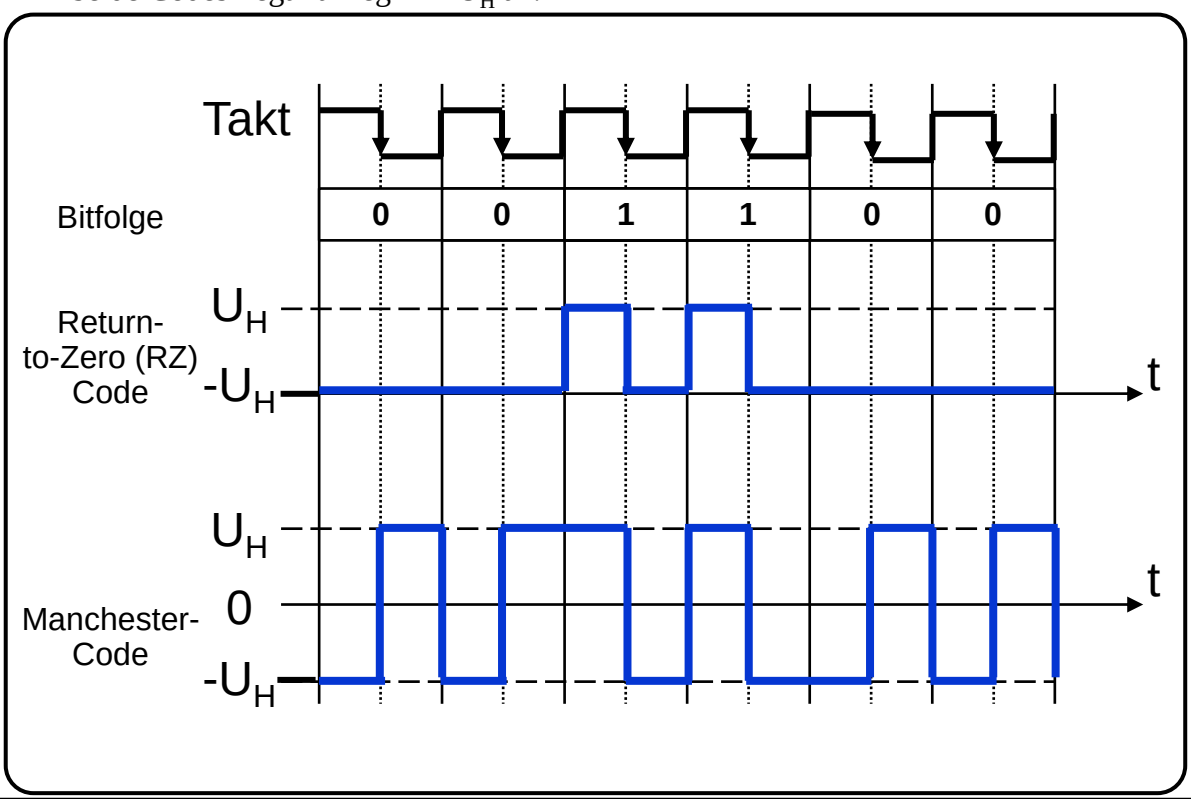
Aufgabe 1: Grundlagen der Informationstechnik und Digitaltechnik

a) Gegeben ist das dargestellte, wert-und zeitkontinuierliche Signal f(t). Zeichnen Sie den wertdiskreten (aber zeitkontinuierlichen) Signalverlauf von f(t) in das Schaubild im Intervall [t0, t7] mit ein. Runden Sie hierbei zur nächstgelegenen **Ganzzahl**.

Hinweis: Bei 0,3 soll auf 0 gerundet werden; bei Werten von 0,5 oder 0,7 auf 1.



b) Sie versenden die Bitfolge `001100` auf einem seriellen Bussystem. Zeichnen Sie den resultierenden Leitungscode als Return-to-Zero-Code und im Manchester-Code. Für beide Codes liegt zu Beginn $-U_H$ an.



Aufgabe 2: IEEE 754 Gleitkommadarstellung und Zahlensysteme

a) Rechnen Sie die gegebene Gleitkommazahl (angelehnt an die IEEE 754 Darstellung) in eine Dezimalzahl um, indem Sie die folgenden Textblöcke ausfüllen.

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

0	1	0	0	0	1	0	1	1	0	0	0
V	E (4 Bit)				M (7 Bit)						

Vorzeichen

0 “positiv” +

Bias als Dezimalzahl

$B = 2^{(x-1)} - 1 = 7$

Biased Exponent als Dezimalzahl

$E = (1000)_2 = (8)_{10}$

Exponent als Dezimalzahl

$e = E - B = 8 - 7 = 1$

Vollständige Dualzahl (denormalisierte Mantisse) ohne Vorzeichen

$= (1{,}1011000)_2 * 2^1 = (11{,}011000)_2$

Vollständige Dezimalzahl (inkl. Vorzeichen)

+ 3,375

b) Überführen Sie die unten gegebenen Zahlen in die jeweils anderen Zahlensysteme.
Hinweis: Achten Sie genau auf die jeweils angegebene Basis.

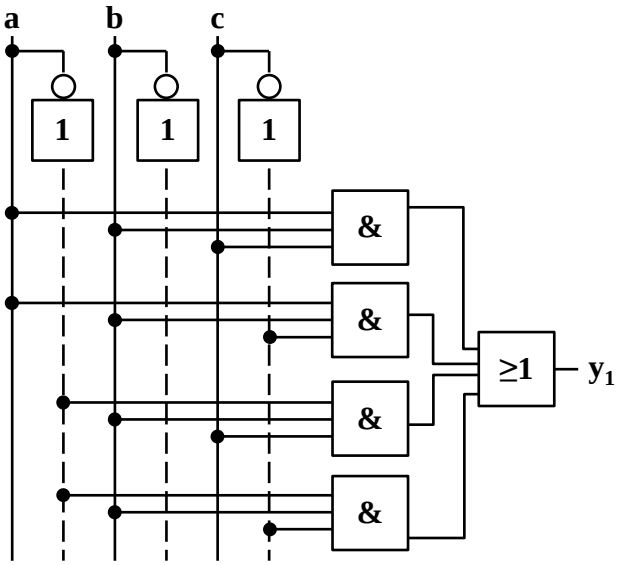
1 **(1 0 0) 3 = (10) 9 = (9) 10**

2 **(42) 10 = (2A) 16**

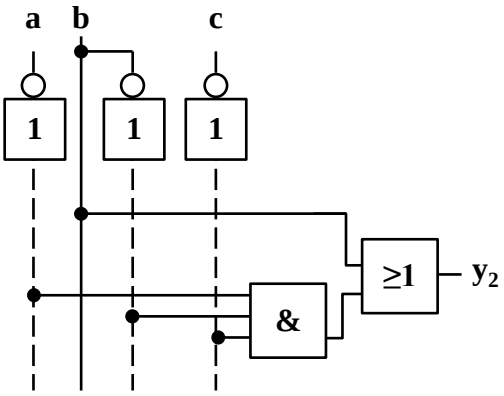
Aufgabe 3: Logische Schaltungen und Schaltbilder

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

Prüfen Sie das Schaltungsverhalten, indem Sie die unten angegebene Wahrheitstabelle (Tabelle 3.1) ausfüllen.



Schaltung 1



Schaltung 2

Tabelle 3.1: *Wahrheitstabelle*

a	b	c	y_1	y_2
1	1	1	1	1
1	1	0	1	1
1	0	1	0	0
1	0	0	0	0
0	1	1	1	1
0	1	0	1	1
0	0	1	0	0
0	0	0	0	1

Aufgabe 4: FlipFlops

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

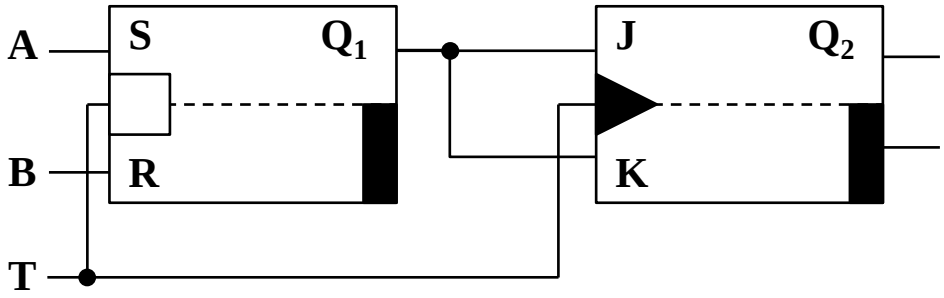
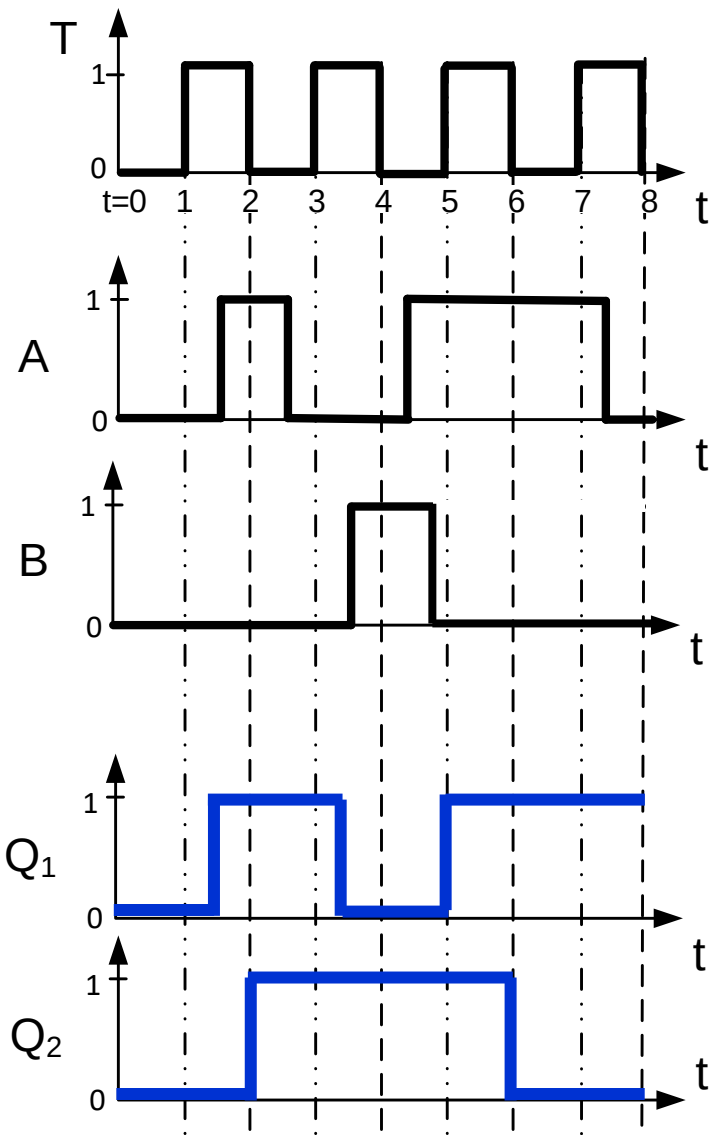


Bild 4.1: *MS-FF*

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

Analysieren Sie die Schaltung, 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.

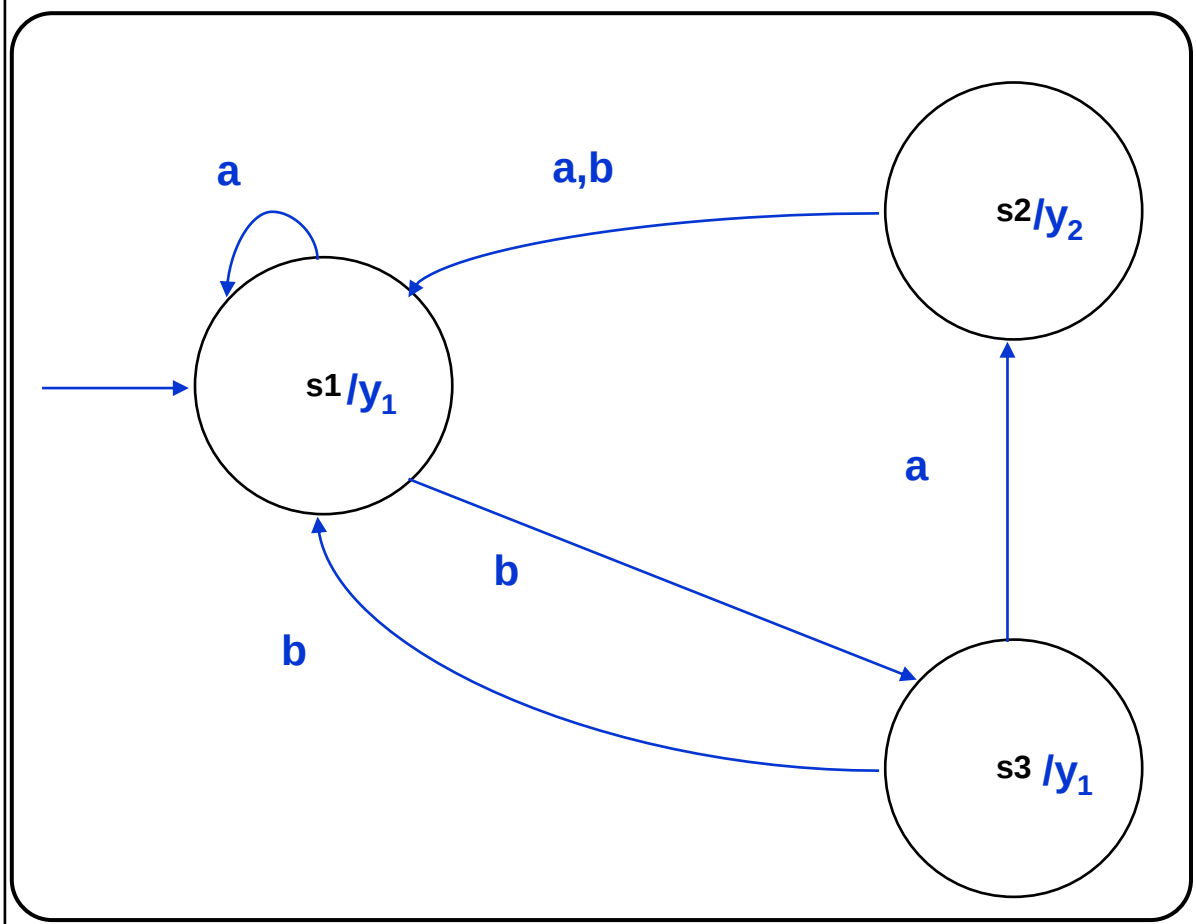


Aufgabe 5: Automaten

Gegeben sei die folgende Übergangstabelle mit den Zuständen s1, s2 und s3, den möglichen Eingaben a und b und den Ausgaben y1, y2.

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

a) Vervollständigen Sie den untenstehenden Automat unter der Annahme, dass s1 der Startzustand ist.



Aufgabe 6: MMIX – Assembler-Code

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: $x = \frac{c}{(a+b)^2 - c} - 16$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x86	0x00 00 00 00 00 00 62 0F	Nicht veränderbar
$0x87	0x00 00 00 00 00 00 AF FE	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 68 72	Zwischen-ergebnis
$0x8B	0x00 00 00 00 00 00 01 00	Nicht veränderbar
…	…	

Bild 6.2: *Registerspeicher*

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 die 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	ADD $0x8A, $0x87 $0x88
2	MUL $0x8A, $0x8A, $0x8A
3	SUB $0x8A, $0x8A, $0x89
4	DIV $0x8A, $0x89, $0x8A
5	SUBI $0x8A, $0x8A, 0x10

Aufgabe 7: MMIX und Betriebssysteme

a) Sie möchten den Wert 0xAFFE (Variable a, Bild 7.1, links) als Wyde an die Speicheradresse 0x0...630F in den Datenspeicher (Bild 7.1, rechts) schreiben.

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
$0x86	0x00 00 00 00 00 00 62 0F	
$0x87	0x00 00 00 00 00 00 AF FE	Variable a
…	…	
$0x8B	0x00 00 00 00 00 00 01 00	

Datenspeicher	
Adresse	Wert
…	…
0x00 00 00 00 00 00 63 0C	0x00
0x00 00 00 00 00 00 63 0D	0x00
0x00 00 00 00 00 00 63 0E	0x00
0x00 00 00 00 00 00 63 0F	0x00
0x00 00 00 00 00 00 63 10	0x00
0x00 00 00 00 00 00 63 11	0x00
…	…

Bild 7.1: *Register- und Datenspeicher*

Geben Sie den dafür benötigten Assembler-Befehl an:

STW $0x87 $0x86 $0x8B

An welcher Adresse im Speicher steht ‘AF‘ nach dem Schreibvorgang?

0x 00 00 00 00 00 00 63 0E

b) Übersetzen Sie den folgenden Befehl aus Maschinensprache (binär) in Maschinensprache (Hexadezimal) und in Assembler-Code.

Maschinensprache (Binär): 1110 0001 1000 1000 0000 1010 1111 0011

Maschinensprache (Hexadezimal): 0xE 1 8 8 0 A F 3

Assemblersprache: SETMH $0x88 0x0AF3

c) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

	wahr	falsch
1. Der logische Adressraum kann größer als der physikalische sein.	(X)	( )
2. Die Segmentierung mit variabler Größe kann unbenutzbar kleine Teile des Speichers verursachen.	(X)	( )

Aufgabe 8: Echtzeitprogrammiersprache PEARL

Vervollständigen Sie den untenstehenden Codeausschnitt in PEARL gemäß der Kommentare über den Lücken.

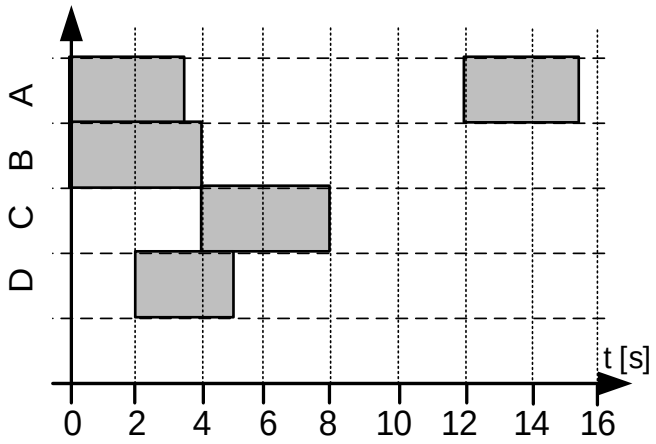
```
// Definieren Sie die Semaphore „Tank“ mit Startwert 1
      DCL Tank SEMA PRESET 1 ;

// Definieren Sie den Task „Leeren“ mit Priorität 4
Leeren : TASK PRIORITY 4 ;

  // Fragen Sie die Semaphore „Tank“ an.
  REQUEST Tank ;

  // Spezifizieren Sie die Unterbrechung „Ueberlauf“.

  SPC Ueberlauf INTERRUPT;

  // Aktivieren Sie die Unterbrechung „Ueberlauf“

  ENABLE Ueberlauf;

  // Aktiviere Task „Nachfuellen“ alle 2 Sekunden

  ALL 2 SEC ACTIVATE Nachfuellen;

  // Wenn die Unterbrechung „Ueberlauf“ auftritt, wird Task
     „Nachfuellen“ blockiert.

  WHEN Ueberlauf SUSPEND Nachfuellen;

  // Geben Sie die Semaphore „Tank“ frei.
  RELEASE Tank ;

  // Ende des Tasks
  END;
```

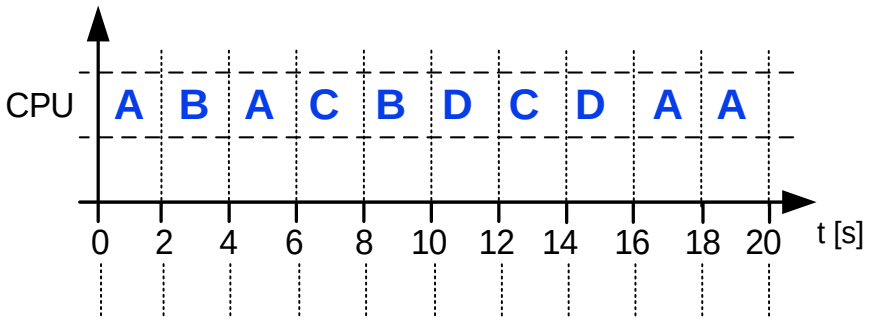
Aufgabe 9: Scheduling und Semaphoren

Gegeben sei der Soll-Verlauf der vier Prozesse A-D (Bild 9.1), welche nach dem Schema Round-Robin (RR) für den Zeitraum $t = [0; 20]$ s und einem Zeitschlitz von zwei Sekunden auf einem Einkernprozessor (CPU) eingeplant werden sollen. Task A und Task D benötigen dieselbe Semaphoren, um auf der Round-Robin-Zeitscheibe **eingeplant** zu werden, d.h. sie können **niemals gleichzeitig auf der Scheibe** liegen..

Geben Sie für alle zwei Sekunden im Zeitraum t = [2;20] s an, in welcher Reihenfolge welche Tasks auf der Round-Robin-Zeitscheibe liegen (z.B. bei t=0: A liegt vor B auf der Zeitscheibe, vgl. erste Spalte in Tabelle in Bild 9.2).

Bild 9.1: *Einplanung / Soll-Verlauf der Tasks*





Tasks auf Round-Robin-Zeitscheibe

A B	B A	A C B	C B D	B D C	D C	C D	D	A	A	-

Bild 9.2: *Ist-Verlauf der Tasks mit Round-Robin*

Achtung: A und D dürfen niemals gleichzeitig auf der Scheibe liegen!

Aufgabe 10: IEC 61131-3 Funktionsbausteinsprache

Nachfolgend soll das Zu- und Abflussverhalten eines Wassertanks entwickelt werden.

1 Das Zuflussventil *V_Zu* öffnet sich (*V_zu* wird *true*), sobald der Tankbodensensor *SensBod* meldet, dass kein Wasser mehr im Tank ist (*SensBod* ist *false*).

V_Zu wird geschlossen, falls der Notausschalter (*Notaus*) aktiviert wird oder der Überlaufsensor *SensUeber* meldet, dass Wasser aus dem Tank überläuft (*SensUeber* ist *true*). Das Schließen des Ventils *V_zu* hat im Zweifel Vorrang.

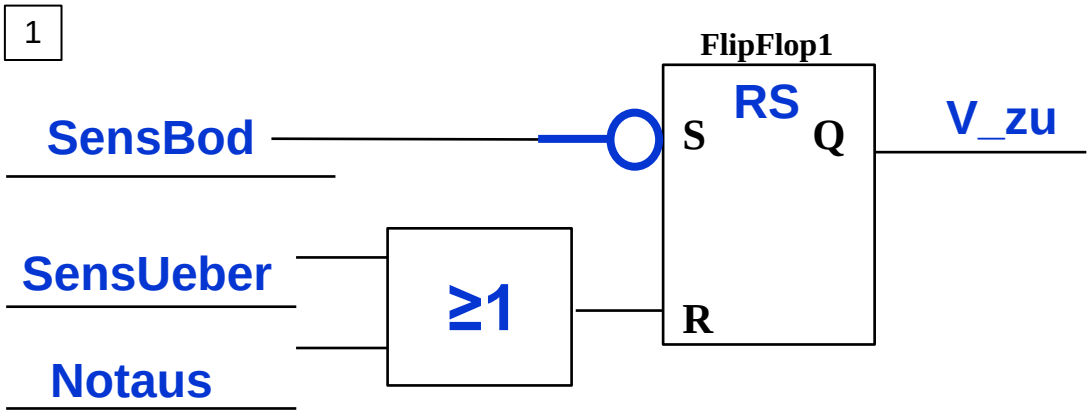
2 Das Abflussventil *V_ab* wird geöffnet, wenn der *Handtaster* nicht betätigt (*Handtaster* ist *false*) wird und gleichzeitig der Tankbodensensor Wasser im Tank detektiert (*SensBod* ist *true*).

Das Ventil schließt (*V_ab* ist *false*) bei einem *Notaus*. Im Zweifel hat aber das Öffnen des Ventils *V_ab* Vorrang.

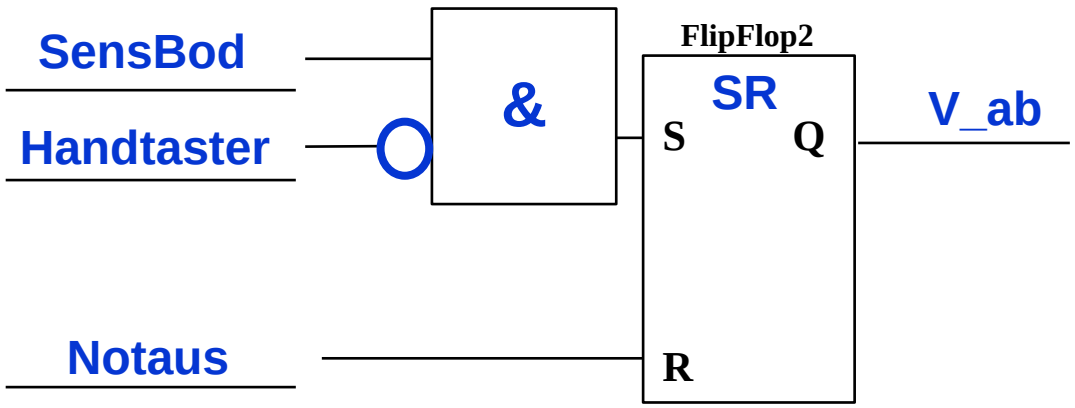
Ergänzen Sie die untenstehenden Programme [1] und [2] so, dass das oben beschriebene Verhalten erfüllt wird.

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

1



2

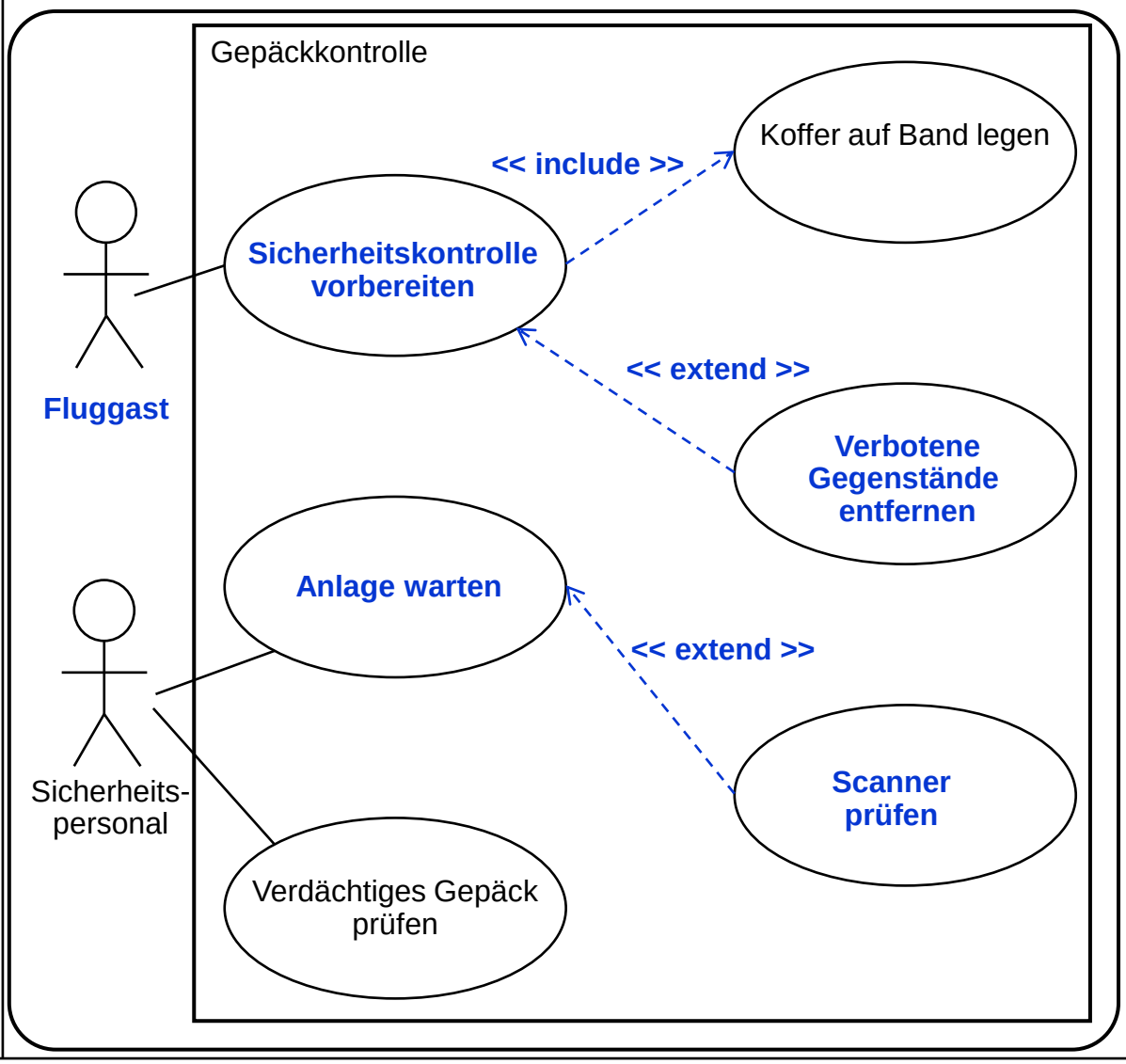


Aufgabe 11: Vervollständigen Sie das UML-Use-Case-Diagramm

Die Gepäckkontrolle an Flughäfen ist ein hochtechnisierter Prozess, der bei allen Beteiligen maximale Sicherheit gewährleisten soll. Um zu garantieren, dass trotz der hohen Software-Komplexität keine Störungen auftreten, sollen Sie in den folgenden Aufgaben eine Anlage zur Gepäckkontrolle mithilfe der UML modellieren und ihre Software in C bzw. C++ implementieren.

Das Gepäckstück wird zunächst vom *Fluggast* zur Sicherheitskontrolle gebracht. Dazu muss dieser die *Sicherheitskontrolle vorbereiten*. Dies beinhaltet den Vorgang „*Koffer auf Band legen*“. Falls im Gepäck verbotene Gegenstände, wie zum Beispiel Flüssigkeiten, enthalten sind, ist zusätzlich der Schritt *verbotene Gegenstände entfernen* notwendig. Meldet die Gepäckkontrolle ein verdächtiges Gepäckstück, muss das *Sicherheitspersonal* dieses prüfen. Zudem ist das *Sicherheitspersonal* für die Wartung der Anlage (*Anlage warten*) zuständig. Je nach Ergebnis der Wartung ist ggf. noch eine zusätzliche, genauere Überprüfung des Scanner nötig (*Scanner prüfen*).

Füllen Sie mithilfe der obigen Angaben das Use-Case-Diagramm der UML für die Gepäckkontrollanlage aus. Benennen Sie die Akteure sowie die Anwendungsfälle. Ergänzen Sie die Verbindungen zwischen den Use-Cases mit richtiger Beziehungsart und Richtung.

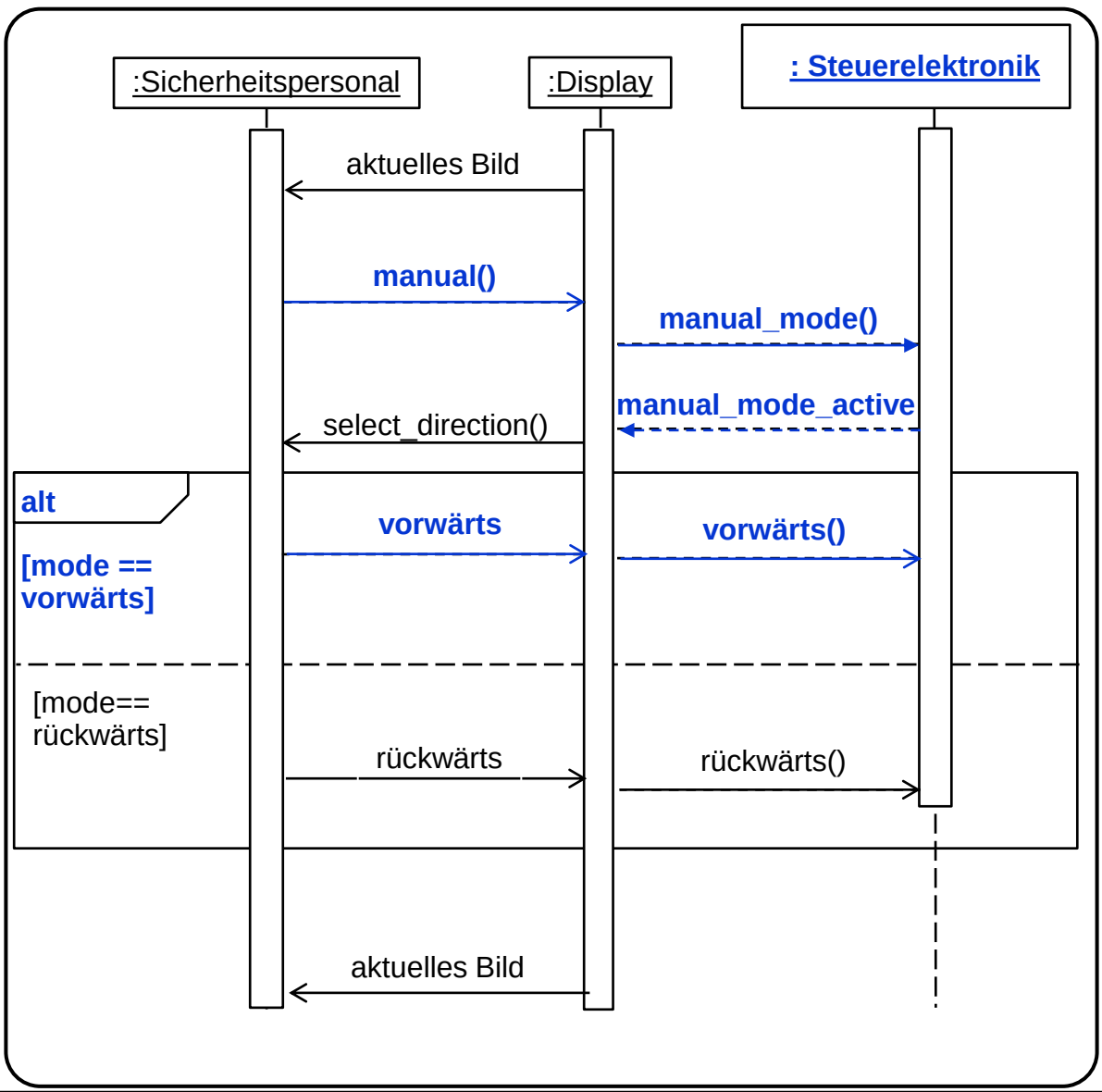


Aufgabe 12: Vervollständigen Sie das UML-Sequenzdiagramm

Das *Display* kommuniziert mit dem *Sicherheitspersonal* ausschließlich über asynchrone Nachrichten. Es zeigt ihm das jeweils aktuelle Bild des Gepäckstücks im Scanner an. Bei verdächtigen Gepäckstücken schaltet das Sicherheitspersonal am Display mit der Methode *manual()* in einen manuellen Modus zur genaueren Überprüfung des Gepäckstücks. Das Display gibt den Methodenaufruf *manual_mode()* weiter an die *Steuerelektronik*. Die Steuerungselektronik bestätigt dem Display die Aktivierung des Modus (*manual_mode_active*). Das Display fordert dann das Sicherheitspersonal auf, die Transportrichtung zu wählen (*select_direction*). Anschließend entscheidet das Sicherheitspersonal, ob das Gepäckstück im Scanner vorwärts (*mode* ist *vorwärts*) oder rückwärts (*mode* ist *rückwärts*) fahren soll. Die jeweilige Entscheidung (*vorwärts* bzw. *rückwärts*) wird vom Sicherheitspersonal an das Display und vom Display durch die Methode *vorwärts()* bzw. *rückwärts()* an die Steuerelektronik weitergegeben. In jedem Fall gibt das Display anschließend wieder das aktuelle Bild des Gepäckstücks im Scanner an das Sicherheitspersonal aus.

Ergänzen Sie das Sequenzdiagramm entsprechend der Beschreibung.

Alle Nachrichten sind mit gestrichelten Linien vorgegeben. Ergänzen Sie die Pfeilspitzen und ändern Sie – falls notwendig – die Linie in eine durchgezogene Linie.

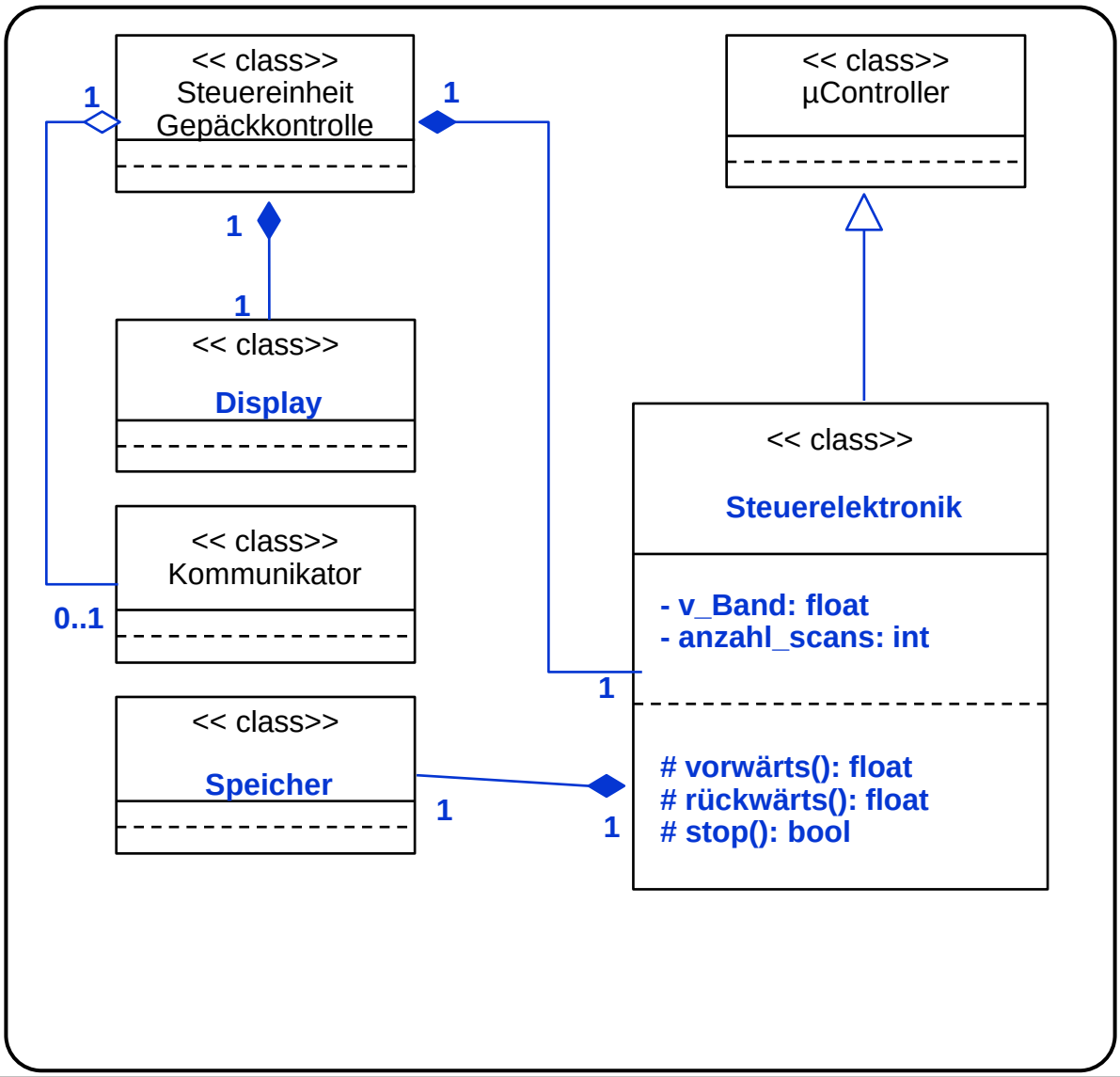


Aufgabe 13: Vervollständigen Sie das UML-Klassendiagramm

Die Steuereinheit der Gepäckkontrollanlage besteht immer aus genau einer *Steuerelektronik* und genau einem *Display*. **Optional** kann die Steuereinheit maximal einen *Kommunikator* beinhalten, um mit weiteren Infrastrukturkomponenten des Flughafens zu kommunizieren.

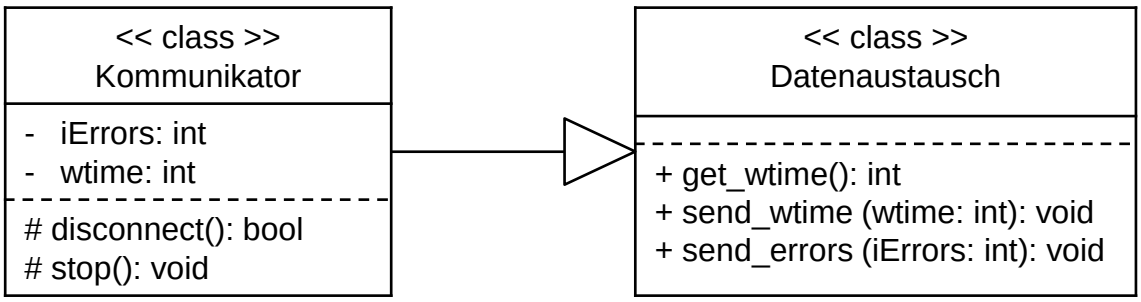
- Die *Steuerelektronik* ist eine Spezialisierung der Klasse *µController*.
- Die *Steuerelektronik* besteht immer aus genau einem *Speicher*.
- Im privaten Attribut *v_Band* der Klasse *Steuerelektronik* ist die jeweils aktuelle Geschwindigkeit der Förderbänder in der Anlage als Fließkommazahl gespeichert. Das ebenfalls private Attribut *anzahl_scans* der Klasse *Steuerelektronik* speichert die Zahl der insgesamt durchgeführten Scans der Anlage als Ganzzahl.
- Die *Steuerelektronik* bietet drei geschützte (protected) Operationen: *vorwärts()* gibt die aktuelle Geschwindigkeit als Fließkommazahl zurück, *rückwärts()* gibt die aktuelle Geschwindigkeit als Fließkommazahl zurück, *stop()* stoppt die Anlage und gibt *true* bei stehenden Förderbändern zurück

Vervollständigen Sie das untenstehende UML-Klassendiagramm um Klassennamen, Attribute, Methoden und Beziehungen. Beachten Sie die Kardinalitäten zu den Beziehungen.



Aufgabe 14: Überführen Sie das UML-Klassendiagramm in C++-Code

Es wird nun die Kommunikation der Gepäckkontrollanlage näher betrachtet. Sie haben das folgende Klassendiagramm gegeben:



Alle benötigten Header-Dateien sind bereits eingebunden.

Geben Sie die Klassendeklarationen der zwei im Klassendiagramm dargestellten Klassen *Datenaustausch* und *Kommunikator* in **C++** an.

Hinweis: Die Anzahl an Linien im Lösungskästchen ist bei allen Programmieraufgaben unabhängig von der Anzahl an geforderten Code-Linien.

```
class Datenaustausch_____________________________{
public:__________________________________________
_________________________________________________
  int get_wtime();_______________________________
  void send_wtime(int wtime);____________________
  void send_errors(int iErrors);_________________
_________________________________________________
_________________________________________________
_________________________________________________
};

class Kommunikator :public Datenaustausch   {
_________________________________________________
private:_________________________________________
  int iErrors;___________________________________
  int wtime;_____________________________________
_________________________________________________
protected:_______________________________________
  bool disconnect();_____________________________
  void stop();___________________________________
_________________________________________________
_________________________________________________
};
```

Aufgabe 15: Grundlagen in C – Array und Zeiger

a) Deklarieren Sie ein eindimensionales Array namens `Buffer` in der Programmiersprache C, in welchem genau 1000 Werte vom Datentyp float gespeichert werden können. Initialisieren Sie alle Elemente des Arrays mit 0.0.

Hinweise:

- Es müssen nur die Anweisungen im Aufgabentext ohne vollständiges C-Programm implementiert werden.
- Die Anzahl an Linien im Lösungskästchen ist bei allen Programmieraufgaben unabhängig von der Anzahl an geforderten Code-Linien.

```
float Buffer[1000] = {0.0};

Alternativ:
float Buffer[1000];
for (int i = 0; i < 1000; i ++){
      Buffer[i] = 0.0;}
```

b) Gegeben ist eine Variable namens `a`, welche wie folgt deklariert und initialisiert wurde:

```
int a = 3;
```

Führen Sie folgende Anweisungen nacheinander in der Programmiersprache C aus. Das Einbinden von Bibliotheken ist **nicht** notwendig.

Deklarieren Sie einen Pointer `pA`, welcher die Adresse der Variablen a speichern kann und initialisieren Sie diesen Pointer `pA` mit der 0-Adresse.

```
int *pA = NULL;
```

Speichern Sie in den Pointer `pA` die Adresse der Variablen `a`.

```
pA = &a;
```

Ändern Sie den Wert der Variablen `a` ausschließlich durch Deferenzierung von `pA` auf den Wert 0.

```
*pA = 0;
```

Aufgabe 16: Grundlagen in C – Schleifen und Funktionspointer

a) Wandeln Sie in der Programmiersprache C die nachfolgende for-Schleife in eine while-Schleife um, welche dieselbe Funktionalität wie die for-Schleife erfüllt.

```
for (int i = 0; i < 10; i -=2){
      printf("%i", i – 2);
}
```

```
int i = 0;
while (i < 10)
{
    printf("%i", i-2);
    i -= 2;
}
```

b) Eine Funktion namens `ais` wurde wie folgt implementiert:

```
void ais (float b, int c){
      printf("%.1f, %.2f", b, (float) c);
}
```

Führen Sie folgende Anweisungen nacheinander in der Programmiersprache C aus. Das Einbinden von Bibliotheken ist **nicht** notwendig.

Deklarieren Sie einen Funktionspointer namens `aispointer`, welcher die Adresse der Funktion `ais` speichern soll. Eine Initialisierung des Pointers ist nicht notwendig.

```
void (*aispointer)(float, int);
```

Weisen Sie dem Funktionspointer die Funktion `ais` zu.

```
aispointer = ais;

Alternativ:   aispointer = &ais;
```

Rufen Sie die Funktion `ais` ausschließlich durch die Verwendung von `aispointer` mit den Inputparametern `b = 1.0, c = 2` auf.

```
aispointer(1.0, 2);

Alternativ:   (*aispointer)(1.0,2);
```

Welche Ausgabe erhalten Sie auf der Kommandozeile nach Ausführen der Funktion `aispointer` mit den Inputparametern `b = 1.0, c = 2`.

```
1.0, 2.00
```

Aufgabe 17: Ergänzen Sie das UML-Zustandsdiagramm

Die bisher betrachtete Gepäckkontrollanlage soll nun genauer modelliert werden. Hierfür wurde zunächst eine schematische Abbildung (**Abb. 17.1**) erstellt. Der Ablauf einer Gepäckkontrolle ist wie folgt gegeben:

Ankommende Gepäckstücke werden von der ersten Lichtschranke (int **LG1* wird 1) erkannt. Daraufhin werden diese so lange transportiert, bis sie den Röntgenscanner erreichen, was durch Lichtschranke 2 (int **LG2* wird 1) erkannt wird. Der Transport erfolgt mittels zweier Förderbandmodule, die jeweils über einen Motor (Motor **iMotor1* bzw. Motor **iMotor2*) angesteuert werden. Motoren werden durch Setzen auf 1 aktiviert und durch Setzen auf 0 deaktiviert bzw. gestoppt.

Der Röntgenscanner gibt nach einer Zeitspanne von 5000ms die Dichte der gemessenen Gepäckstücke als Integer (*iErgebnis*) zurück. Liegt der Wert unter oder gleich 400, wird von keiner Gefahr ausgegangen. Das Tor öffnet sich (Methode *TorHeben*) und das Gepäckstück wird weitertransportiert (**iMotor2 = 1*), bis Lichtschranke 3 (int **LG3*) erreicht ist. Das Tor wird anschließend wieder geschlossen (Methode *TorSenken*) und das Förderbandmodul 2 abgeschaltet (**iMotor2 = 0*). Bei einem Dichte-Wert von größer 400 bleibt das Tor geschlossen und *Sicherheitspersonal* wird angefordert. Nach manueller Inspektion und ggfs. Entfernen des Gepäckstücks muss die Anlage wieder freigegeben werden. Hierfür drückt das *Sicherheitspersonal* einen Quittierknopf (*bQuittieren* wird *TRUE*). Die Anlage wartet anschließend wieder auf ankommende Gepäckstücke. Bei einem Fehler (*ifehler == 1)* im Zustand *Wartend* wird das Programm der Anlage beendet.

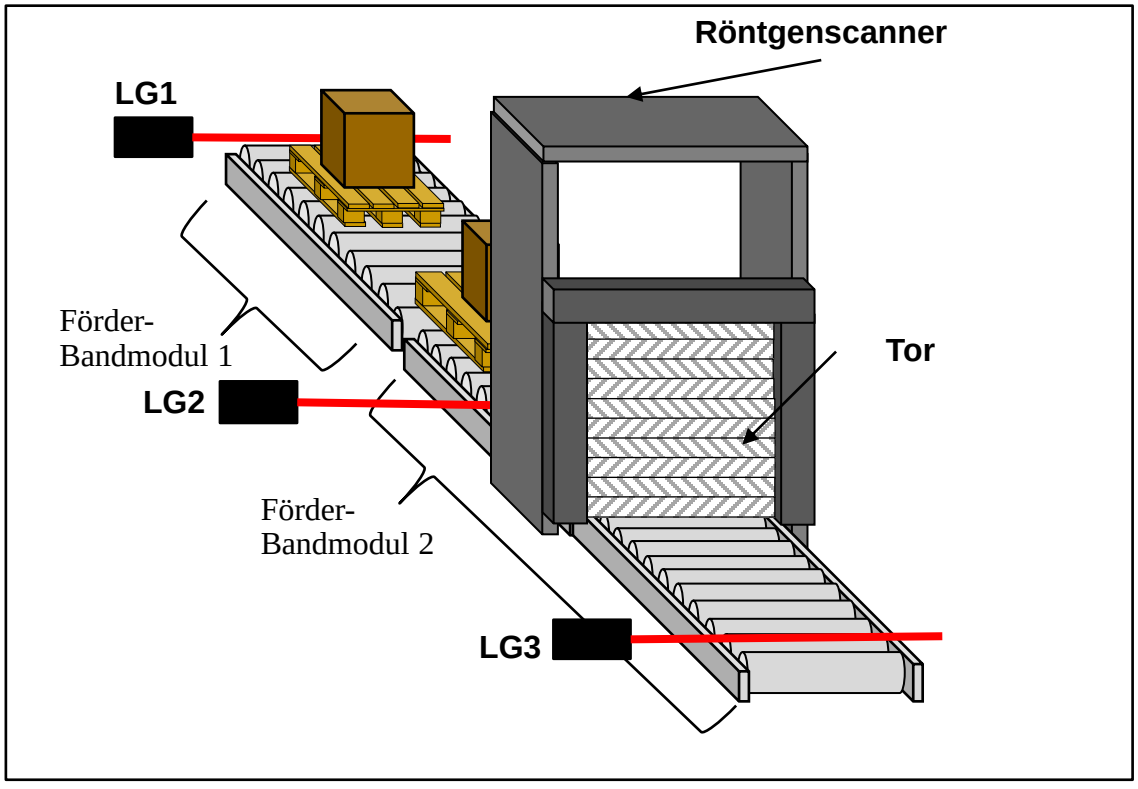


Abb. 17.1 – Sensorik und Aktorik der Gepäckkontrollanlage

Es ist das in **Abb. 17.2** gezeigte Zustandsdiagramm mit den Zustandsnummern 1 bis 5 gegeben. Füllen Sie die durch römische Ziffern gekennzeichneten Lücken im Lösungsfeld aus, bzw. beantworten Sie die gegebenen Fragen.

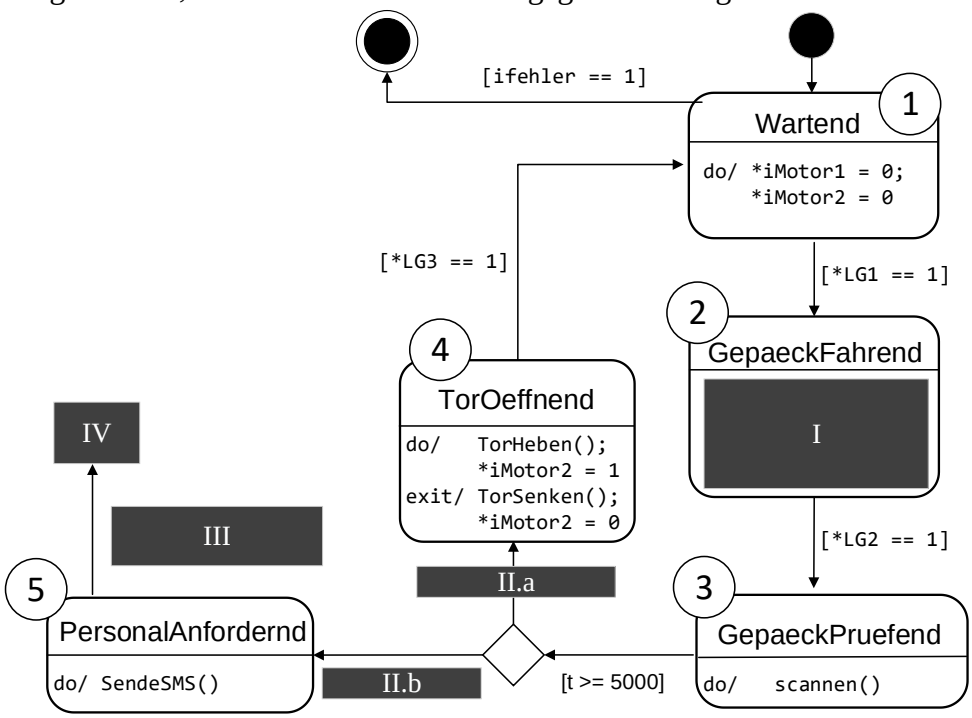


Abb. 17.2 – Zustandsdiagramm der Betriebssoftware der Gepäckkontrollanlage

I. Modellieren Sie den Zustand 2 „`GepaeckFahrend`“ korrekt aus:

do/ *iMotor1 = 1;
*iMotor2 = 1
exit/ *iMotor1 = 0;
*iMotor2 = 0

II. Geben sie die korrekte Wächterbedingung an:

II.a [iErgebnis <= 400]

II.b [iErgebnis > 400]

III. Geben sie die korrekte Wächterbedingung bei III an:

[bQuittieren == TRUE] alternativ: [bQuittieren]

IV. Welcher Zustand muss nach dem Zustand 5 „`PersonalAnfordernd`“ folgen, um den Ablauf, wie in der Beschreibung erläutert, zu gewährleisten.

1: Wartend

Aufgabe 18: UML-Zustandsdiagramm zu C-Code

Implementieren Sie Teile des in Aufgabe 17 modellierten Zustandsdiagramms mittels C-Code. Nutzen Sie hierfür die in **Tabelle 18.1** dargestellten Variablen sowie die bereits implementierte Funktion `initsystem` zur Initialisierung des Systems.

Tabelle 18.1 – Sensor- und Aktorvariablen, sowie vorgegebene Variablen und bereits implementierte Funktionen

Type	Name	Beschreibung
VARIABLEN	unsigned int *vplcZeit	Pointer für aktuelle Zeit der SPS in ms ab SPS Start
	MOTOR *iMotor1	Pointer für den Motortreiber-Baustein von Motor 1
	MOTOR *iMotor2	Pointer für den Motortreiber-Baustein von Motor 2
	int *LG1	Pointer für Lichtschranke 1
	int *LG2	Pointer für Lichtschranke 2
	int *LG3	Pointer für Lichtschranke 3
	int iZustand	Schrittvariable des Zustandsautomaten
	unsigned int t	Initiale Zeitvariable
	int iErgebnis	Ergebnis des Scanvorgangs
FUNKTIONEN	void initsystem(unsigend int *m_Zeit, int *m_Sensor1, int *m_Sensor2, int *m_Sensor3, MOTOR *m_Motor1, MOTOR * m_Motor2)	Initialisierungsfunktion des Systems: Initialisierung der Pointer für SPS-Zeit, Sensorik und Aktorik
	int scannen()	Gibt die Dichte des Gepäckstücks zurück
	int wiegen()	Gibt das Gewicht des Gepäckstücks in kg zurück

a) Vervollständigen Sie das im folgenden Lösungskästchen gezeigte Programmgerüst gemäß der Kommentare in der Programmiersprache C. Verwenden Sie hierfür die in **Tabelle 18.1** angegebenen Variablennamen sowie die gegebene Funktion, welche im Header `gepaeckanlage.h` der Gepäckkontrollanlage bereits definiert und implementiert ist.

Hinweis: Die Platzhalter /* ZUSTAENDE */ enthalten spezifischen Code für Zustände des Zustandsautomaten und müssen nicht implementiert werden.

```
// Header der Gepäckkontrollanlange einbinden
#include „gepaeckanlage.h“  alternativ: <> statt ““
// Pointer fuer die Lichtschranken gemäß Tab.18.1 deklarieren
int *LG1;
int *LG2;
int *LG3;
// Variablen fuer Motoren deklarieren
MOTOR *iMotor1;
MOTOR *iMotor2;

// Pointer fuer aktuelle Zeit der SPS gemäß Tab.18.1 deklarieren
unsigned int *vplcZeit;
int main()
{   // Deklarationen
    int iZustand = 1;        //Startzustand
    unsigned int t = 0;      //initiale Zeitvariable
    iErgebnis = 0;           //Ergebnis des Scanvorgangs

    // Initialisierung des Systems
    initsytem(vplcZeit, LG1, LG2, LG3, iMotor1, iMotor2);

    while(1)                 //Zyklische Ausfuehrung (endlos)
    {
        switch(iZustand)     //Zustandsautomat
        {
            /*    ZUSTAENDE   */
        }
    }
   return 0;
}
```

Nach einem Umbau der Gepäckkontrollanlage wird diese nun zum Wiegen der Gepäckstücke eingesetzt. Der Röntgenscanner wurde durch eine Waage ersetzt.
Im Zustand *GepaeckPruefend* wird deshalb nun das Gewicht des jeweiligen Gepäcks ermittelt. Hierfür wird, anders als zuvor, zyklisch die Funktion *wiegen()* aufgerufen, welche das Gewicht in kg in der bereits deklarierten Variable *iErgebnis* speichert. Für ein präzises Messergebnis muss das Gepäckstück mindestens 5 Sekunden auf der Waage verbleiben. Nach Ablauf dieser Zeit wird gemäß **Abb. 18.1** bei akzeptablem Gewicht entweder der Zustand 4 (*TorOeffnend*) oder bei Schwerlast der Zustand 5 (*PersonalAnfordernd*) erreicht. Ein Gewicht von kleiner gleich 23 kg wird als akzeptabel betrachtet. Bei Gepäck über 23kg handelt es sich um Schwerlast (vgl. **Abb. 18.1**).

Hinweis: Verwenden Sie einen Timer, um die Wartezeit von 5000ms auf der Waage zu realisieren. Gehen Sie davon aus, dass der Timer beim erstmaligen Eintritt in den Zustand den Wert t=0 hat. Setzen Sie t beim Verlassen des Zustandes *GepaeckPruefend* zurück.

b) Implementieren Sie den Zustand *GepaeckPruefend* der Anlage gemäß obiger Beschreibung und **Abb. 18.1** in der Programmiersprache C. Verwenden Sie zudem die angegebenen Variablennamen aus Tabelle 18.1.

```
case 3:     //Zustand GepaeckPruefend
iErgebnis = wiegen();
if(t == 0)
{
     t = *vplcZeit + 5000;
{
else
{
    if (t <= *vplcZeit)
    {
        t = 0;
        if (iErgebnis <= 23)
        {
            iZustand = 4;
        }
        else
        {
            iZustand = 5;
        }
    }
}


break;
```

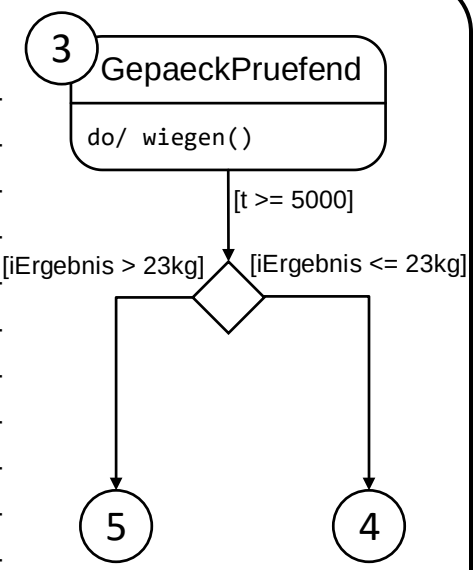


Abb. 18.1: Zustände 3, 4 und 5

Aufgabe 19: Algorithmen und Datenstrukturen

Die Gepäckkontrollanlage soll nun aus einzelnen Förderbandmodulen (vgl. **Abb. 19.1**) aufgebaut werden. Die Förderbandmodule werden nacheinander in Linie zusammengefügt. Eine Eigenschaft, anhand derer die Förderbandmodule beschrieben werden können, sind deren 2-dimensionale Koordinaten im Raum.

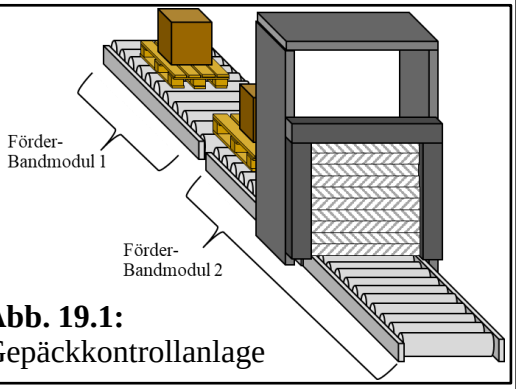


Abb. 19.1:
Gepäckkontrollanlage

- x-Koordinate des Förderbandmoduls (`x`) als Fließkommazahl
- y-Koordinate des Förderbandmoduls (`y`) als Fließkommazahl

a) Implementieren Sie in der Programmiersprache C ein `struct` namens `Koordinaten` mit den folgenden Elementen:

- x-Koordinate des Förderbandmoduls (`xdim`) als Fließkommazahl
- y-Koordinate des Förderbandmoduls (`ydim`) als Fließkommazahl

Hinweise:
- Die Anzahl an Linien im Lösungskästchen ist bei allen Programmieraufgaben unabhängig von der Anzahl an geforderten Code-Linien.
- Es müssen keine Header eingebunden werden.

```
struct Koordinaten{
    float xdim;
    float ydim;
};
```

Um die Planung der Gepäckkontrollanlage zu vereinfachen, soll eine **einfach verkettete Liste** erstellt werden, sodass die Eigenschaften jedes Förderbandmoduls einem Listenelement eineindeutig zugeordnet werden können. Die Konfiguration der Gepäckkontrollanlage soll im Listenkopf abgespeichert werden. Dem Programm liegt ein 64-Bit-Betriebssystem zugrunde (vgl. **Tabelle 19.1**).

In einem Listenelement vom Typ `MODUL` sollen gespeichert werden:
- Ein Zeiger (`pnext`), welcher auf das nächste Listenelement vom Typ `MODUL` zeigt
- Die Position (`position`) im struct-Datentyp `Koordinaten`.
- Die Anbindungsinformation des Fördermoduls in einem Array namens `information` der Länge 2 als 32-Bit vorzeichenlose Ganzzahl.

In dem Listenkopf vom Typ `MODULCONFIG` sollen gespeichert werden:
- Ein Zeiger (`pfirst`), welcher auf das erste Listenelement vom Typ `MODUL` zeigt.
- Die Anzahl an angebundenen Förderbandmodulen als vorzeichenlose 8-Bit Ganzzahl namens `anzahl`.
- Der Konfigurationsname (`name`). Der `name` besteht aus exakt zehn Zeichen.

Hinweise:
- Eine Implementierung des struct-Datentypen `Koordinaten` erfolgt in Aufgabe 19 a) und ist hier nicht gefordert.
- Berücksichtigen Sie das abschließende Nullzeichen bei Zeichenketten (z.B. `name`).
- Wählen Sie die Größe Ihrer Datentypen so, dass möglichst wenig Speicherplatz belegt wird.

Tabelle 19.1: Datentypgrößen des 64-Bit-Betriebssystems

Datentyp	char	short	int	long	long long	void*
Bits pro Datentyp	8	16	32	64	64	64

b) Vervollständigen Sie die nachfolgenden Lösungskästchen zur Definition von Listenkopf und Listenelement unter Verwendung der Programmiersprache C.

```
typedef struct modul{
    unsigned int information[2];
    struct Koordinaten position;
    struct modul *pnext;
} MODUL;
```

```
typedef struct {
    char name[11];
    char anzahl;
    MODUL *pfirst;
} MODULCONFIG;
```

Implementieren Sie eine Funktion mit dem Namen `erzeugen`, welche den Konfigurationsnamen (`name`) sowie die Anzahl an angebundenen Modulen (`anzahl`) aus einem Listenkopf vom Typ `MODULCONFIG` (siehe Aufgabe 19b) liest. Die Funktion `erzeugen` soll eine Datei mit dem Konfigurationsnamen (`name`) sowie der angefügten Endung „`.txt`“ erstellen und die Anzahl der angebundenen Förderbandmodule im Format „`module: [Anzahl]`“ in der Datei speichern:

Beispiel::

```
Konfigurationsname: abcdefghij
Dateiname: abcdefghij.txt
Anzahl an angebundenen Module: 3
Inhalt der Datei: module: 3
```

Beispiel für die Datei „abcdefghij.txt“:

```
module: 3
```

Die Funktion `erzeugen` soll als Übergabeparameter einen Pointer auf den Listenkopf erhalten. Die Datei soll im Mode write/translate geschrieben werden.

Hinweis: In `string.h` wird nachfolgende, nützliche Funktion beschrieben:

```
char *strcat( char *dest, const char *src );
```

Beschreibung: Fügt zwei Zeichenketten aneinander, indem es eine Kopie der Byte-Zeichenkette (nullterminiert), auf die `src` zeigt, an das Ende der Byte-Zeichenkette (nullterminiert), auf die `dest` zeigt, anhängt. Das Zeichen `src[0]` ersetzt das Nullzeichen am Ende von `dest`. Die resultierende Byte-Zeichenkette ist nullterminiert.

Parameter:

dest: Zeiger auf die Byte-Zeichenkette (nullterminiert), die **vorne** stehen soll

src: Zeiger auf die Byte-Zeichenkette (nullterminiert), die **hinten** angefügt werden soll.

c) Implementieren Sie die Funktion `erzeugen` in der Programmiersprache C. Es müssen keine Header eingebunden werden.

```
void erzeugen (MODULCONFIG *data)
{
FILE* pDB = fopen(strcat(data->name,“.txt“),“wt“);
fprintf(pDB, “module: %i“, data->anzahl);
fclose(pDB);
}
```

d) Implementieren Sie in der Programmiersprache C eine Funktion mit dem Namen `anbinden`, welche ein Listenelement `MODUL` (siehe Aufgabe 19b) an das Ende der verketteten Liste einfügen und die in dem einzufügendem Listenelement enthaltenen Engineering-Informationen initialisieren soll. Die Funktion `anbinden` soll als Übergabeparameter einen Pointer auf den Listenkopf (`data`) sowie die x- und y-Koordinaten des Förderbandmoduls (`float x, float y`) übergeben bekommen.

Die in dem Listenelement enthaltenen Engineering-Informationen sollen wie folgt initialisiert werden:

- Die x-Koordinate (`xdim`) der Position (`position`) mit dem Übergabeparameter (`x`).
- Die y-Koordinate (`ydim`) der Position (`position`) mit dem Übergabeparameter (`y`).
- Das erste Feld des Arrays (`information`) enthält eine binäre Flag, welche auf `True` gesetzt werden soll (Element wurde erfolgreich angebunden).
- Das letzte Feld des Arrays (`information`) enthält die Nummer, an welcher Stelle das Listenelement eingefügt wurde. Das erste Listenelement enthält die Nummer 1.

Hinweis: Das letzte Listenelement zeigt immer auf NULL.

```
void anbinden (MODULCONFIG *data, float x, float y)
{
// Zaehlvariable i fuer Position des Listenelem.
     int i = 1;
// Deklarierung des einzufuegenden Moduls
     MODUL* newmodul;
// Zeiger auf erstes Listenelement
     MODUL* pointer = data->pfirst;

_____________________________________________
_____________________________________________
while(pointer!= NULL)
   {
       pointer = pointer->pnext;
       i++;
   }

   newmodul = (MODUL*) malloc(sizeof(MODUL));
   pointer = newmodul;
   newmodul->pnext = NULL;

   newmodul->position.xdim = x;
   newmodul->position.ydim = y;
   newmodul->information[0] = 1;
   newmodul->information[1] = i;

}
```

Innerhalb der `main()`-Funktion des Programmes soll eine Funktionalität zur Planung der Gepäckkontrollanlage entworfen werden. Die Planung der Anlage beginnt mit der Erstellung des Listenkopfes `MODULCONFIG` (siehe Aufgabe 19b).

e) Implementieren Sie in der Programmiersprache C folgende Schritte in der `main()`, indem Sie das Lösungskästchen geeignet ergänzen:

- Deklarieren Sie für den Listenkopf eine Variable (`anlage`) vom Typ `MODULCONFIG`.
- Lesen Sie den durch den Benutzer über die Kommandozeile eingegebenen Konfigurationsnamen ein.
- Initialisieren Sie den Listenkopf mit geeigneten Werten. Gehen Sie davon aus, dass noch keine Fördermodule angebunden wurden.

Hinweis: Es müssen keine weiteren Header-Dateien eingebunden werden.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    MODULCONFIG anlage;

    printf("Eingabe Konfigurationsname:");
    scanf("%10s", anlage.name );

    anlage.pfirst = NULL;
    anlage.anzahl = 0;


// Hier folgt weiterer Code
```

Zusätzlicher Lösungskasten. Falls Sie den Lösungskasten nutzen, geben Sie bitte die Nummer der Aufgabe an, die Sie hier lösen möchten.