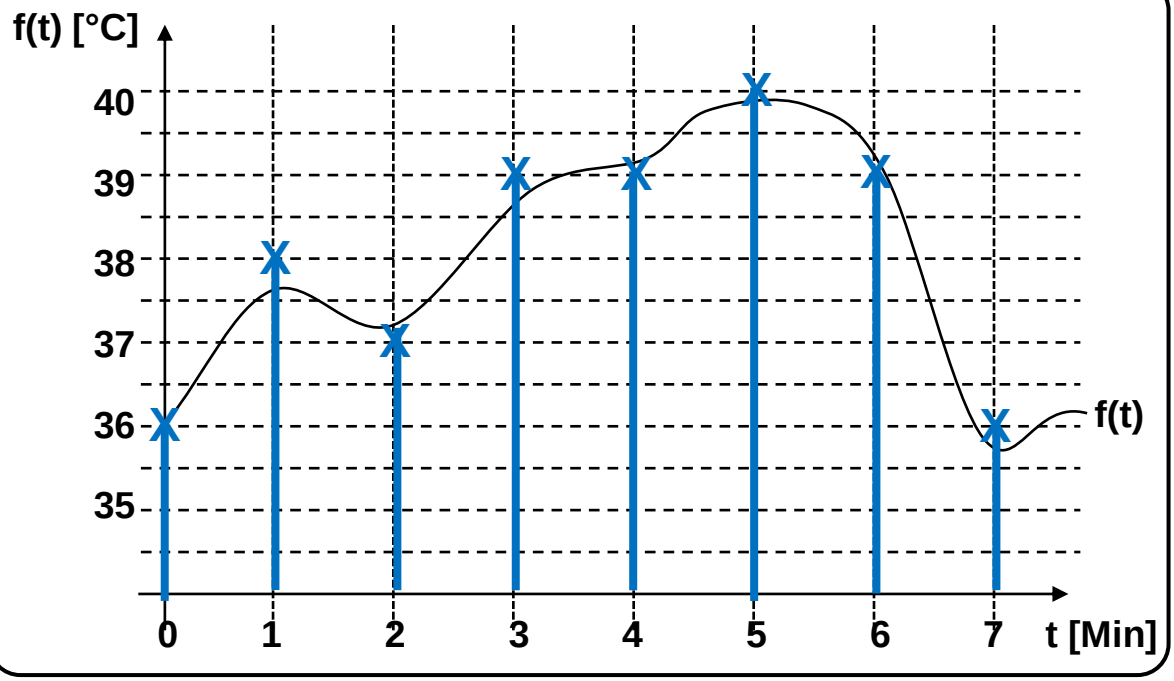
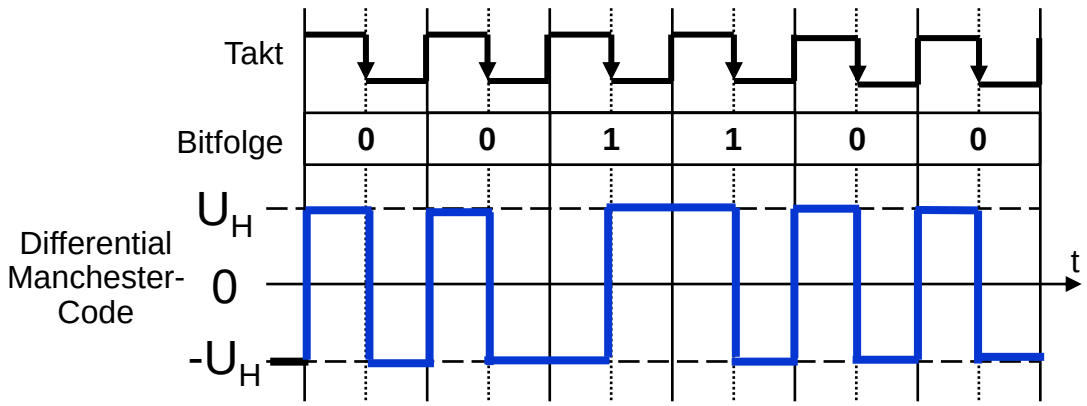
Aufgabe 1: Grundlagen der Informationstechnik und Digitaltechnik

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 Minute mit Rundung auf ganzzahlige °C.



b) Zeichnen Sie den Leitungscode für die Bitfolge `001100` als Differential Manchester-Code. Zu Beginn liegt $-U_H$ an.



c) Sender und Empfänger haben als Basis für ihre Kommunikation die folgenden zwei Codewort-Duos (C1, C2) zur Auswahl. Bestimmen Sie je Duo die Hamming-Distanz sowie die Anzahl behebbarer Fehler.

	C1:	C2:
	0 0 1 1 0 0 1 1 1 0 0 1	0 0 1 1 0 0 1 1 1 1 0 1
Hamming-Distanz:	h = 4	h = 3
Anzahl behebbare Fehler:	f = (h-1)/2 = **1.5 = 1**	f = (h-1)/2 = **1**

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!

1	0	1	1	0	0	0	0	0
V	E (5 Bit)					M (3 Bit)		

Vorzeichen

1 “negativ” -

Bias als Dezimalzahl

$B = 2^{(x-1)} - 1 = 2^{(5-1)} - 1 = 15$

Biased Exponent als Dezimalzahl

$E = (01100)_2 = (12)_{10}$

Exponent als Dezimalzahl

e = E – B = 12 – 15 = -3

Vollständige Dualzahl (denormalisierte Mantisse) ohne Vorzeichen

$M = (1{,}0000)_2 * 2^{-3} = (0{,}001)_2$

Vollständige Dezimalzahl (inkl. Vorzeichen)

- 0,125

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

1 **(0 1 1 1 1 0)** $_2$ **= (** 36 **)** $_8$ **= (** 1E **)**$_{16}$

2 **(1 1 0 1 0)** $_2$ **= (** 26 **)** $_{10}$

Aufgabe 3: Logische Schaltungen und Schaltbilder

a) Vervollständigen Sie die Wahrheitstabelle (Tabelle 3.1) für die Schaltung (Bild 3.1).

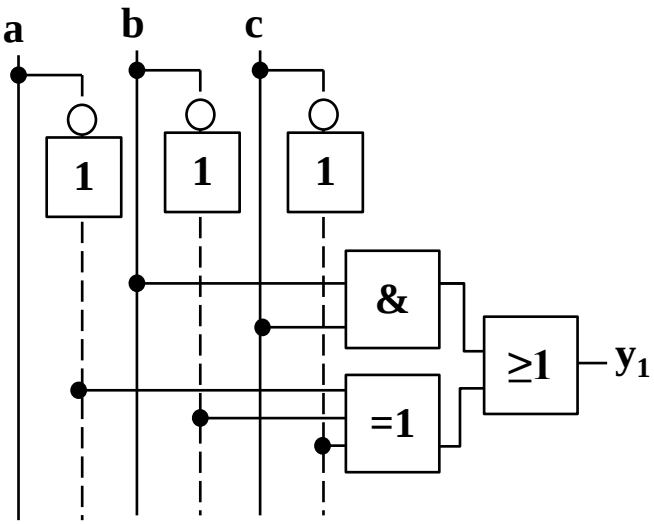


Bild 3.1: *Schaltung*

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

Tabelle 3.1: *Wahrheitstabelle zu Bild 3.1*

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

Hinweis: Die Schreibweise $a \wedge b$ können Sie mit ab abkürzen.

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

Alt.:
$y1 = (\bar{a}\, b\, c) \vee (\bar{a}\, b \bar{c}) \vee (\bar{a}\, \bar{b}\, c)$

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

Tabelle 3.2:
Wahrheitstabelle für DNF

Aufgabe 4: FlipFlops

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

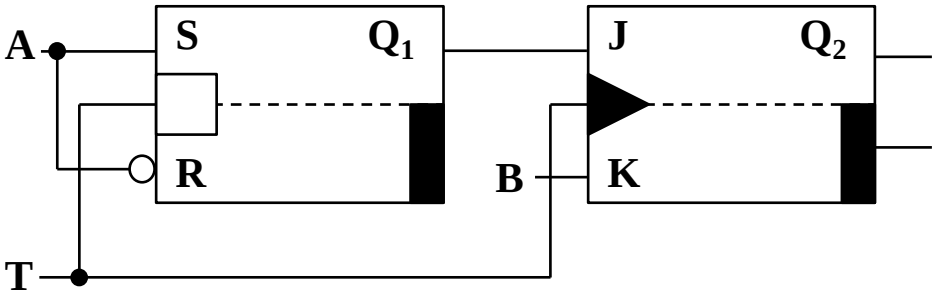
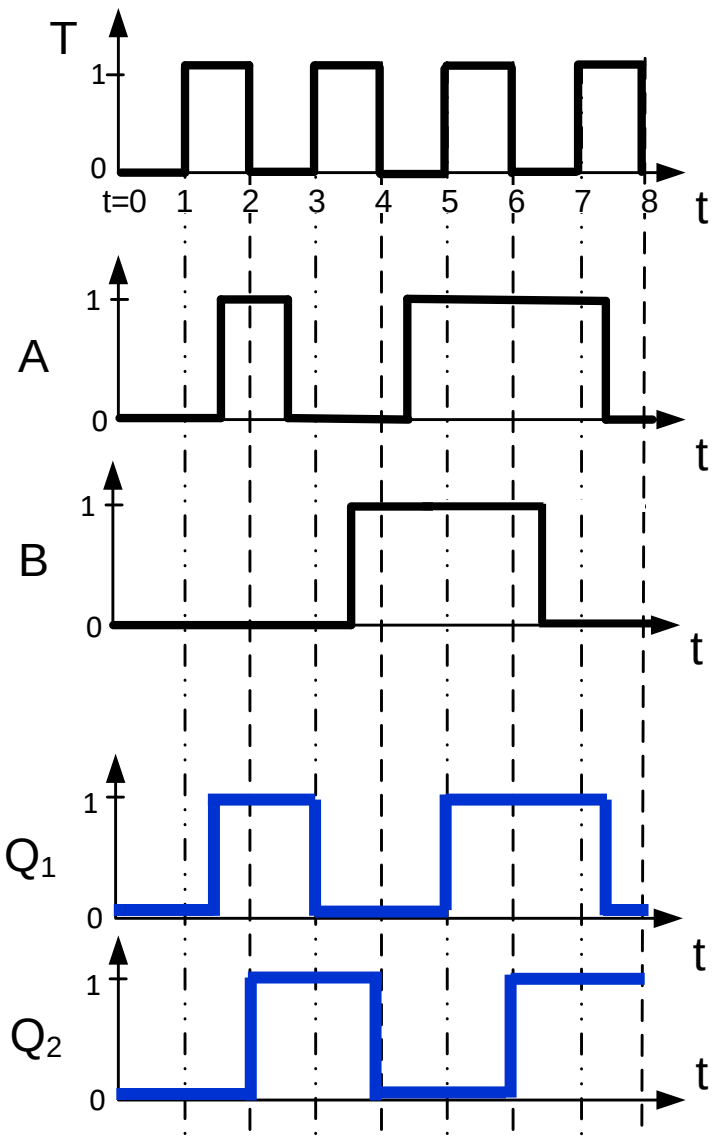


Bild 4.1: *MS-FF*

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: MMIX – Assembler-Code

Gegeben sei der nachfolgende Algorithmus sowie ein Ausschnitt der MMIX-Code-Tabelle (Bild 5.1) und eines Registerspeichers (Bild 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	

Bild 5.1: *MMIX-Code-Tabelle*

Algorithmus: $x = \frac{a}{(a-b)*c} + 18$

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
…	…	…
$0x86	0x00 00 00 00 00 00 62 0F	Zwischen-ergebnis
$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	Endergebnis 1
$0x8B	0x00 00 00 00 00 00 01 00	Endergebnis 2
…	…	

Bild 5.2: *Registerspeicher*

Im Registerspeicher eines MMIX-Rechners befinden sich zu Beginn die in Bild 5.2 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. Verwenden Sie dazu lediglich die in Bild 5.1 **umrahmten Befehlsbereiche**. **Speichern Sie die Zwischenergebnisse** nach jedem Befehl des Algorithmus in der Registerzelle mit dem Kommentar ***Zwischenergebnis***. Übersetzen Sie die Operationen in **Assembler-Code mit insgesamt maximal 4 Anweisungen**.

Nach Ausführen des kompletten Algorithmus soll das finale Ergebnis als **Octa** in den Datenspeicher gespeichert werden. Die Speicheradresse im Datenspeicher ergibt sich aus der Summe der Registerzellen mit dem Kommentar ***Endergebnis 1*** und ***Endergebnis 2***. Geben Sie den Assembler-Code an (Zeile 5).

```
1  SUB $0x86, $0x87 $0x88
2  MUL $0x86, $0x86, $0x89
3  DIV $0x86, $0x87, $0x86
4  ADD I  $0x86, $0x86, 0x12
5  STO $0x86, $0x8A, $0x8B
```

Aufgabe 6: MMIX – Assembler-Code

a) Sie möchten als Wert der Variable a (Bild 6.1, links) ein **Tetra** von der Speicheradresse 0x0...0003 aus dem Datenspeicher (Bild 6.1, rechts) laden. Nutzen Sie dazu **ausschließlich** die Werte im Registerspeicher (Bild 6.1, links).

Registerspeicher		
Adresse	Wert vor Befehlsausführung	Kommentar
$0x86	0x00 00 00 00 00 00 00 01	Wert 1
$0x87	0x00 00 00 00 00 00 CA FE	Variable a
…	…	
$0x8B	0x00 00 00 00 00 00 00 04	Wert 2

Datenspeicher	
Adresse	Wert
…	…
M[0x00 … 00 03]	0x00
M[0x00 … 00 04]	0x04
M[0x00 … 00 05]	0x0D
M[0x00 … 00 06]	0x0E
M[0x00 … 00 07]	0xF0
M[0x00 … 00 08]	0x00
…	…

Bild 6.1: *Register- und Datenspeicher*

Geben Sie den dafür benötigten Assembler-Befehl an:

LDT $0x87 $0x86 $0x8B

Ab welcher Adresse wird aus dem Datenspeicher (Big Endian) gelesen?

M[0x00 00 00 00 00 00 00 04]

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

Assemblersprache: LDB I $0x88 $0x89 0x03

Maschinensprache (Hexadezimal): 0x81 88 89 03

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

	Wahr	Falsch
1. Der Registerspeicher dient der langfristigen Ablage von Daten.	( )	(X)
2. Die Speicherkapazität für einen Wert im Datenspeicher ist kleiner als im Registerspeicher.	(X)	( )

Aufgabe 7: Automaten und Buszugriffsverfahren

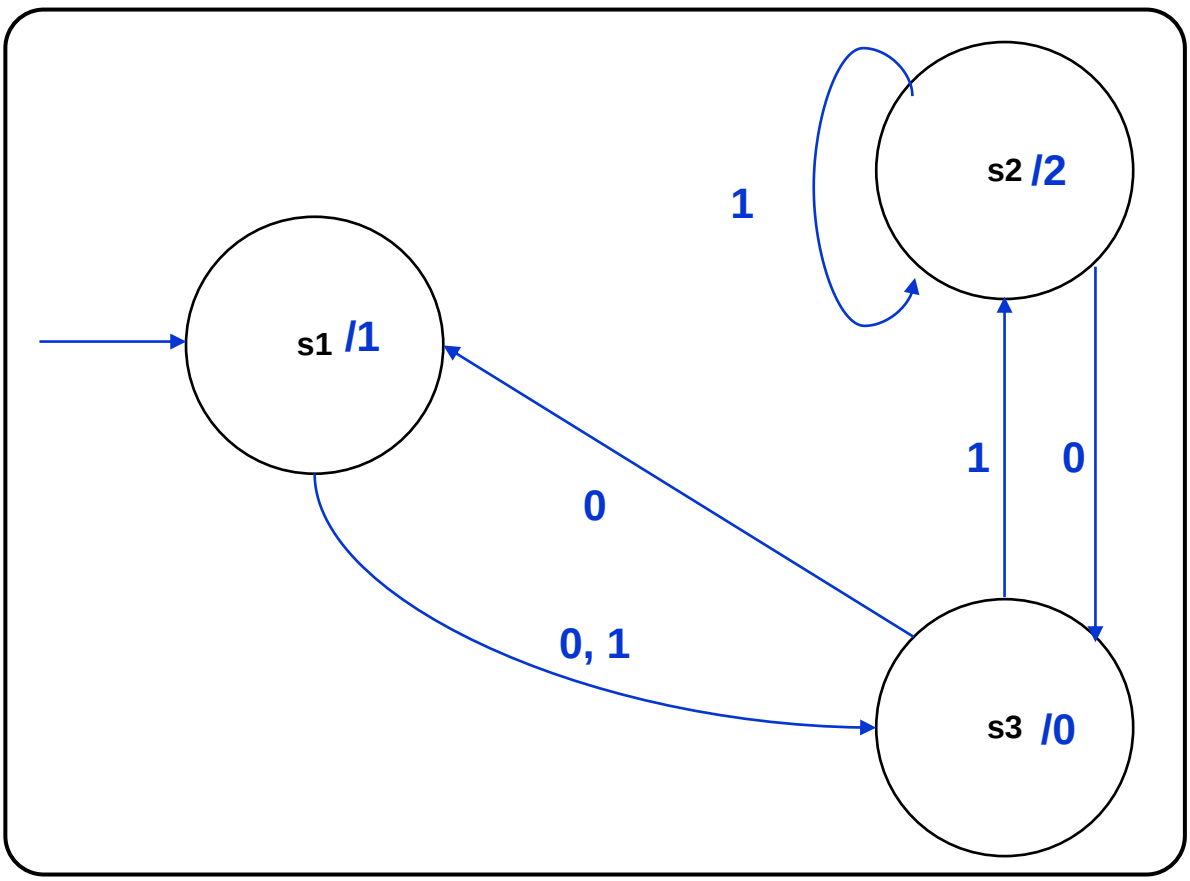
Gegeben sei die folgende Übergangstabelle mit den Zuständen s1, s2 und s3, den möglichen Eingaben 0 und 1 und den Ausgaben 0, 1, 2.

S / T	s1, 1	s2, 2	s3, 0
0	s3	s3	s1
1	s3	s2	s2

a) Sie befinden sich in Zustand s3. Mit welcher Eingabesequenz erhalten Sie die Ausgabe „ 2 2 0 1 “.

1 1 0 0

b) Überführen Sie die Übergangstabelle in den zugehörigen Automat. Startzustand ist s1.



Aufgabe 8: Echtzeitprogrammiersprache PEARL

Vervollständigen Sie den untenstehenden Codeausschnitt einer Anlage zum Befüllen von Flaschen in PEARL gemäß den Kommentaren über den Lücken.

```
// Definieren Sie die Semaphore „Fuellstation“ mit Startwert 1
_____DCL Fuellstation SEMA PRESET 1_____ ;
// Definieren Sie den Task „Foerdern“ mit Priorität 2
Foerdern : _____TASK PRIORITY 2_____ ;
END;


// Task „Fuellvorgang“ für die Ablaufsteuerung.
Fuellvorgang: TASK;
    // Fragen Sie die Semaphore „Fuellstation“ an.
    _____REQUEST Fuellstation_____;
    // Definieren Sie eine Wiederholung, die den Task „Fuellen“
       aktiviert, solange der „FUELLSTAND“ kleiner als 2 ist.
        _____WHILE_____ FUELLSTAND < 2;
        _____REPEAT_____ _____ACTIVATE_____ Fuellen;
        END;
    // Geben Sie die Semaphore „Fuellstation“ frei.
        _____RELEASE_____ Fuellstation;
    // Der Tasks „Foerdern“ wird aktiviert.
    ACTIVATE Foerdern;
 // Beenden Sie den Task „Foerdern“, wenn eine Flasche die
 Lichtschranke „LICHTSENSOR“ passiert („LICHTSENSOR >1“).
 IF LICHTSENSOR >1 THEN_____TERMINATE_____ Foerdern ;
 FIN;
END;
```

Aufgabe 9: IEC 61131-3 Funktionsbausteinsprache

Nachfolgend wird die Bedienung eines industriellen Mahlwerks beschrieben.

1

- Das Bedienpanel verfügt über einen Schutztaster, einen Aktivierungsschalter sowie einen Notaus-Schalter.
- Um den Motor des Mahlwerks (*Motor*) zu starten, müssen sowohl der Aktivierungstaster (*AktTaster*) als auch der Schutztaster (*Schutz*) gedrückt werden. Des Weiteren wird zum Mahlen Material in der Zufuhr benötigt. Ein Sensor (*SensMat*) gibt an ob Material zugeführt wird (*1*) oder nicht (*0*). Ein Notaus-Schalter (*Notaus*) stoppt den Motor. Für den Fall eines Kabelbruchs zwischen Motor und Notaus-Schalter, soll der Motor ebenfalls stoppen, weshalb der Wert von *Notaus* bei Drücken des Schalters 0 ist. *Notaus* hat immer Vorrang.

2

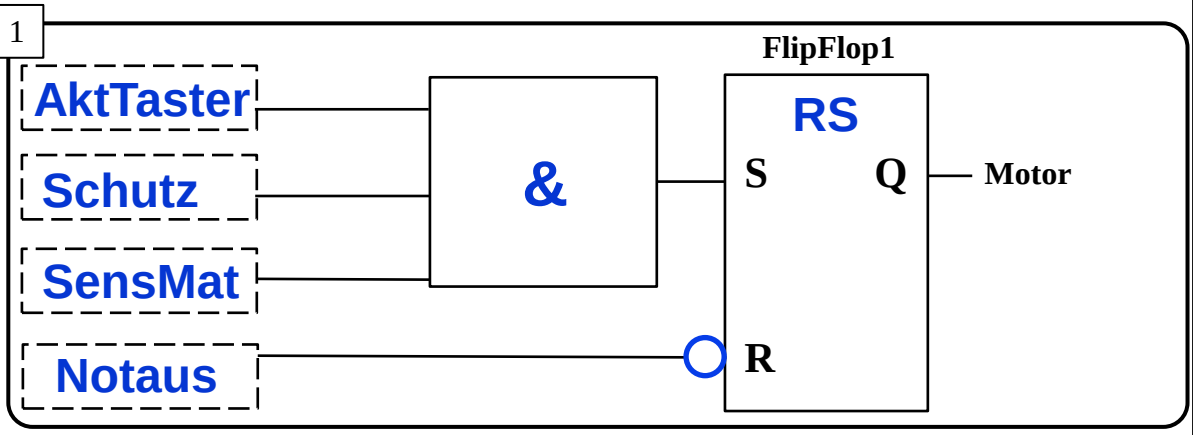
- Zum Schutz des Bedieners gibt es zudem eine Lichtschranke (*SensLicht*), die den Wert 1 annimmt wenn der Lichtstrahl durchbrochen wird. Um ein fehlerhaftes Auslösen zu verhindern, soll dies nur geschehen wenn eine Änderung des Sensorwerts vorliegt. In diesem Fall oder wenn der Notaus (*Notaus*) (0 im Fehlerfall) betätigt wird stoppt die Materialzufuhr (*StopZu*). Das Stoppen hat immer Vorrang.
- Um die Materialzufuhr zu starten, muss der Ausgang *StopZu* über die Taste *Rück* zurückgesetzt werden. In diesem Fall hat *Rück* den Wert 1.

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

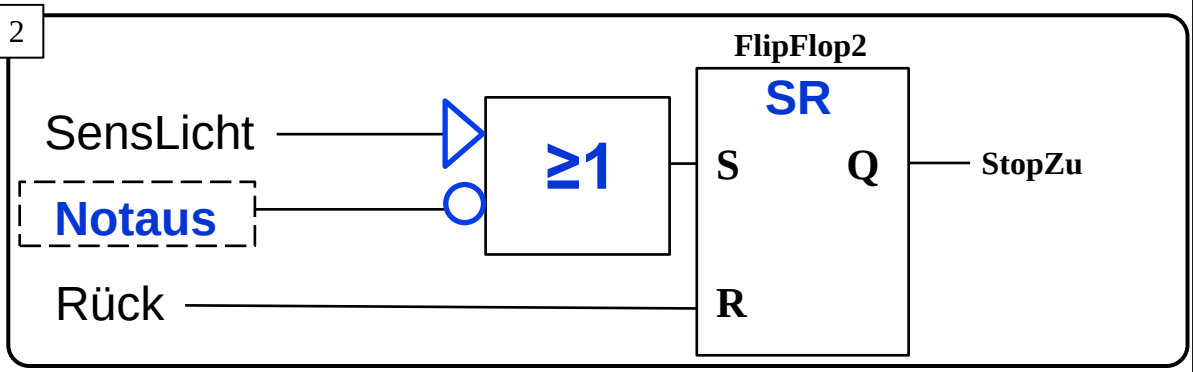
Hinweise:

- 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 10: Scheduling und Semaphoren

Gegeben sei der Soll-Verlauf der vier Prozesse A-D (Bild 9.1), welche nach **festen Prioritäten (s. Bild 9.2) präemptiv** für den Zeitraum $t = [0; 20]$ s und einem Zeitschlitz von zwei Sekunden auf einem Einkernprozessor (CPU) eingeplant werden sollen. Beachten Sie dass neue Tasks immer zum Taktwechsel alle zwei Sekunden eingeplant werden. **Wenn mehrere Tasks die gleiche Priorität haben soll das Round-Robin-Verfahren (RR)** angewandt werden.

Geben Sie für alle zwei Sekunden im Zeitraum t = [2;20] s an, **in welcher Reihenfolge welche Tasks auf der Round-Robin-Zeitscheibe liegen** (Bild 9.3 unten), z.B. bei t=0: A liegt vor B auf der Zeitscheibe (vgl. erste Spalte in Tabelle in Bild 9.3).

Geben Sie zudem die **Reihenfolge,** in der die Tasks auf der CPU bearbeitet werden, an (Bild 9.3 oben).

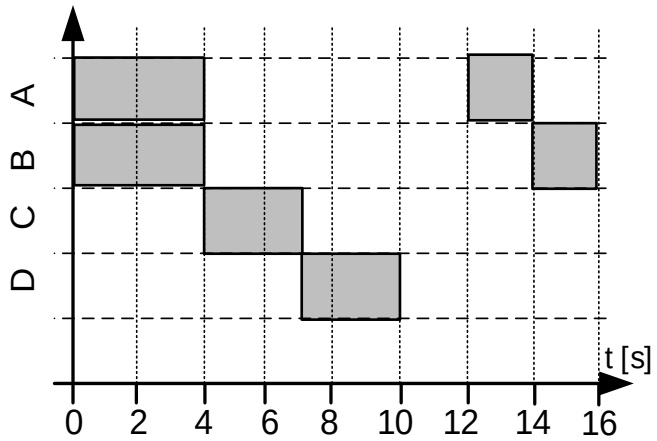
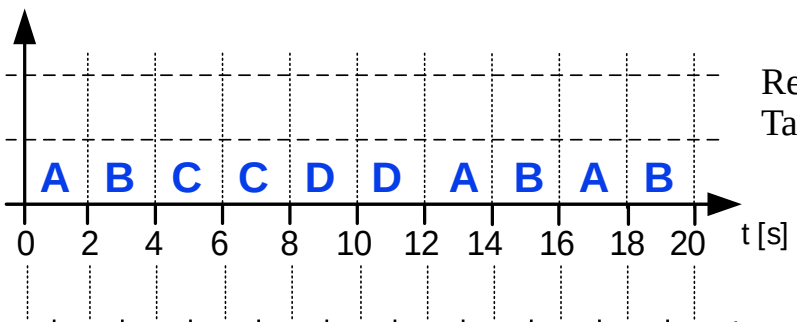


Bild 9.1: *Einplanung / Soll-Verlauf der Tasks*

Tasks	Priorität
A	3
B	3
C	1
D	2

1 ist höchste Priorität

Bild 9.2: *Prioritäten*



Reihenfolge der Tasks auf der CPU

1	A	B	C	C	D	D	A	B	A	B	
2	B	A	A	A	A	A	B	A	B	A	
3			B	B	B	B					
4											
5											

Tasks auf Round-Robin-Zeitscheibe (für Tasks mit gleicher Priorität)

Anmerkung: Hier ist es in sowohl iO wenn nur die Tasks A und B auf die Scheibe gelegt werden als auch wenn jemand alle Tasks einträgt.

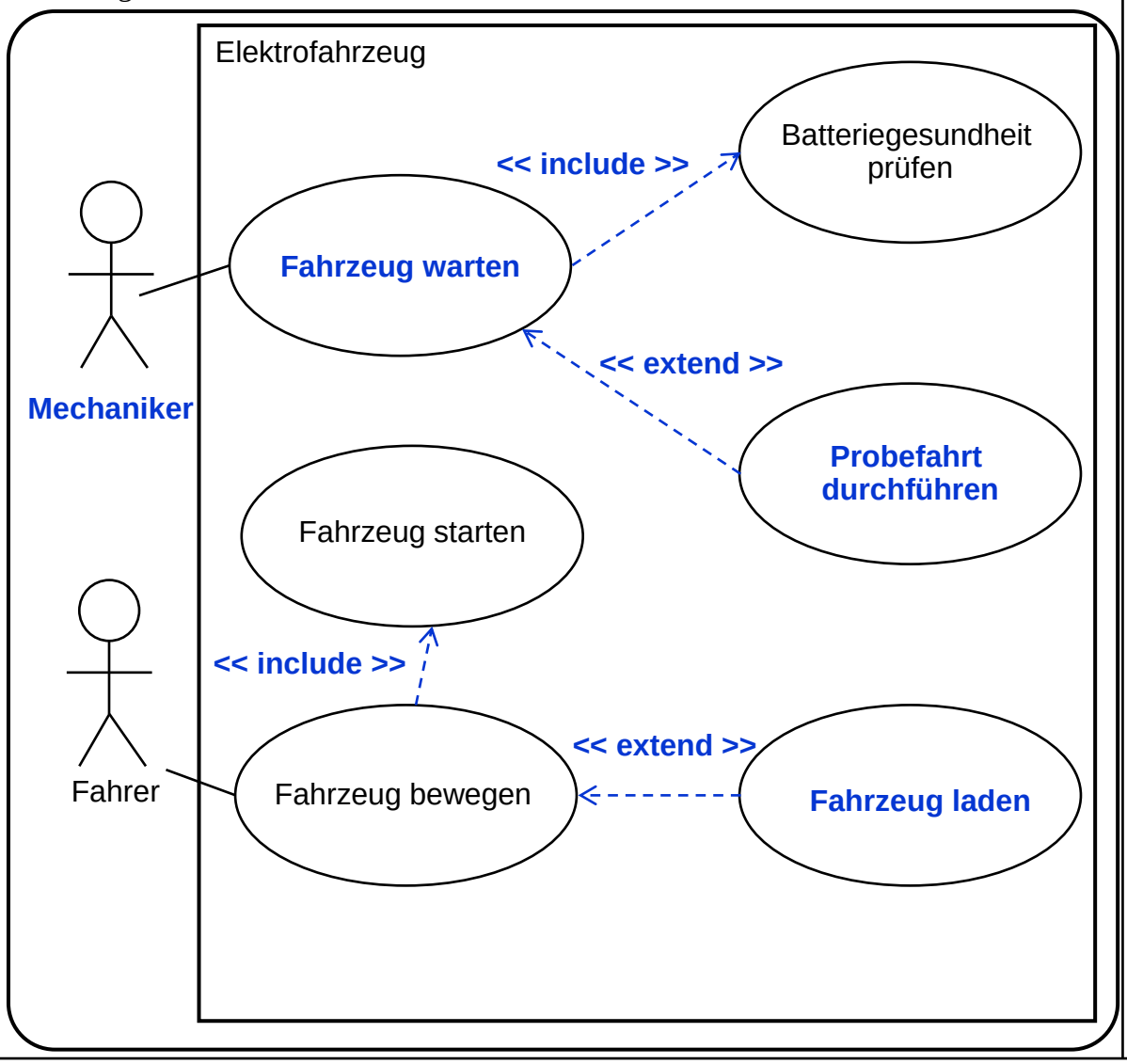
Bild 9.3: *Ist-Verlauf der Tasks mit Prioritäten und Round-Robin*

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

Bei der Entwicklung eines Elektrofahrzeugs ist die maximale Sicherheit im Betrieb oberstes Gebot. Um zu garantieren, dass trotz der hohen Software-Komplexität keine Störungen auftreten, sollen Sie in den folgenden Aufgaben ein Elektrofahrzeug mithilfe der UML modellieren und dessen Software in C bzw. C++ implementieren.

Das Elektrofahrzeug kann vom *Fahrer* bewegt werden (*Fahrzeug bewegen*). Dazu muss dieser das *Fahrzeug starten*. Falls der Ladezustand des Fahrzeugs zu niedrig ist, ist zusätzlich der Schritt *Fahrzeug laden* erforderlich. Zudem muss das Fahrzeug in regelmäßigen Abständen durch einen *Mechaniker* gewartet werden (*Fahrzeug warten*). Eine Wartung schließt hierbei immer die Überprüfung der Batteriegesundheit (State-of-Health) mit ein *(Batteriegesundheit prüfen)*. Je nach Ergebnis der Wartung ist gegebenenfalls noch die Durchführung einer zusätzlichen Probefahrt nötig (*Probefahrt durchführen*).

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

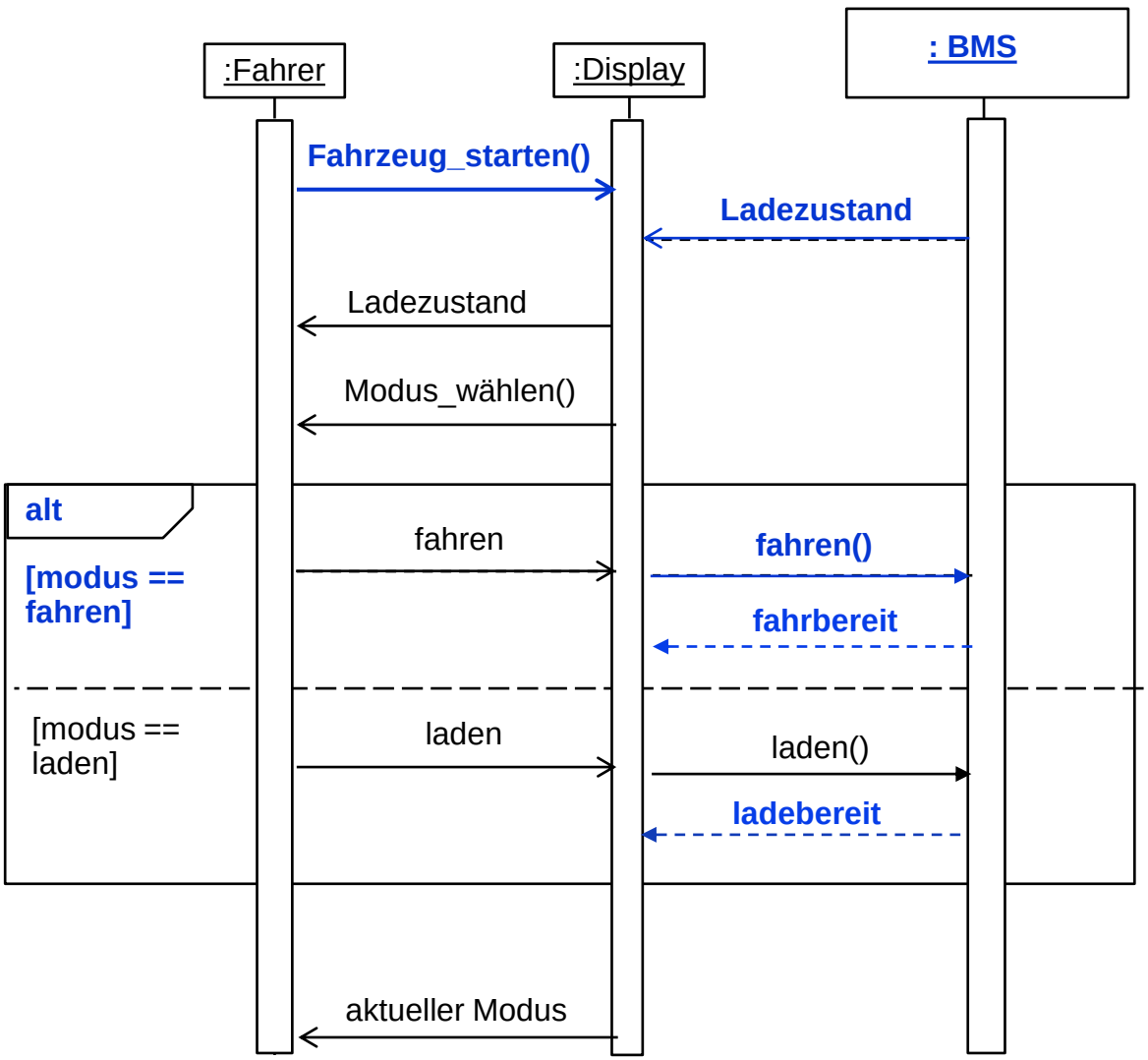


Aufgabe 12: Vervollständigen Sie das UML-Sequenzdiagramm

Der *Fahrer* kann die Fahrzeugsysteme über das Display des Fahrzeugs mit der Methode *Fahrzeug_starten* starten. Das *Display* kommuniziert dabei mit dem *Fahrer* ausschließlich über asynchrone Nachrichten. Anschließend übermittelt das Batteriemanagementsystem (*BMS*) den Ladezustand der Batterie an das Display mittels einer asynchronen Nachricht. Das Display zeigt den *Ladezustand* als Nachricht dem Fahrer an. Anschließend fordert das Display den Fahrer zur Wahl eines Betriebsmodus durch die Methode *Modus_wählen* auf. Der Fahrer entscheidet, ob das Fahrzeug *fahren* (*modus* ist *fahren*) oder *laden* (*modus* ist *laden*) soll. Die jeweilige Entscheidung (*fahren* bzw. *laden*) wird dann vom Display zum BMS durch die Methoden *fahren* bzw. *laden* synchron übermittelt. Anschließend bestätigt das BMS dem Display die erfolgreiche Aktivierung des jeweiligen Zustands (*fahrbereit* bzw. *ladebereit*). Das Display zeigt den Modus (*aktueller Modus*) schließlich dem Fahrer an.

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.

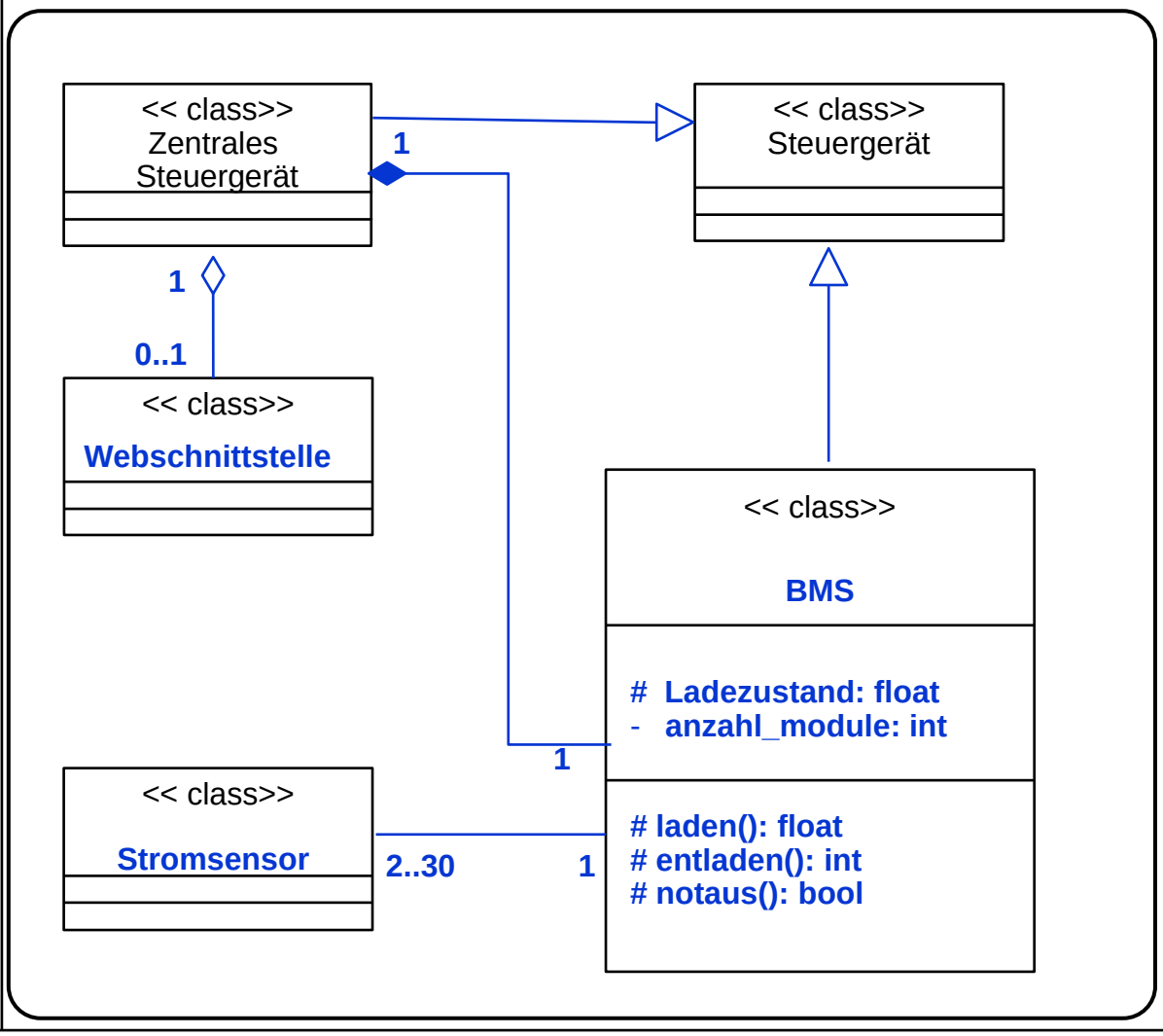


Aufgabe 13: Vervollständigen Sie das UML-Klassendiagramm

Das *zentrale Steuergerät* des Elektrofahrzeugs besteht immer aus genau einem *BMS* und verfügt zudem über maximal eine **optionale** *Webschnittstelle*, um mit dem Server des Fahrzeugherstellers zu kommunizieren. Ein *BMS* ist immer genau einem *zentralen Steuergerät* zugeordnet.

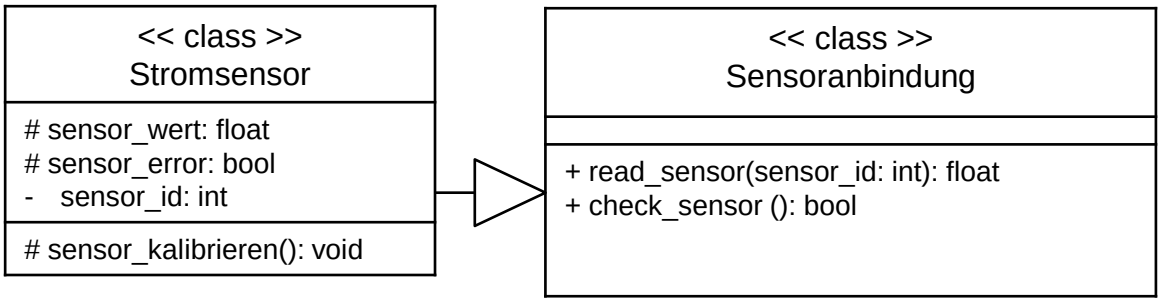
- Das *zentrale Steuergerät* und das *BMS* erben die Eigenschaften der Klasse *Steuergerät*.
- Ein *BMS* kommuniziert mit mindestens zwei und bis zu 30 *Stromsensoren*. Ein Stromsensor kommuniziert immer mit genau einem BMS.
- Im geschützten (protected) Attribut *Ladezustand* der Klasse *BMS* ist der jeweils aktuelle Ladezustand der Batterie als Fließkommazahl gespeichert. Das private Attribut *anzahl_module* der Klasse *BMS* speichert die Zahl der insgesamt in der Batterie vorhandenen Batteriemodule als Ganzzahl.
- Das *BMS* bietet drei geschützte (protected) Operationen: *laden* gibt die aktuelle Ladegeschwindigkeit in kW als Fließkommazahl zurück, *entladen* gibt den aktuellen Ladezustand in Prozent als Ganzzahl zurück, *notaus* trennt die Batterie vom Fahrzeugsystem bei einem Unfall und gibt *true* bei erfolgreicher Trennung 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 der Datenaustausch inkl. Kalibrierung der verbauten Stromsensoren 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 *Sensoranbindung* und *Stromsensor* in **C++** an.

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

```
class Sensoranbindung                    {
public:
  float read_sensor(int sensor_id);
  bool check_sensor();
};

class Stromsensor:public Sensoranbindung   {
private:
  int sensor_id;

protected:
  float sensor_wert;
  bool sensor_error;
  void sensor_kalibrieren();
};
```

Aufgabe 15: Grundlagen in C – Datentypen und Kontrollstrukturen

a) Für diese Aufgabe ist ein Ausschnitt aus einer ASCII-Tabelle (Tabelle 15.1) gegeben. Des Weiteren sind folgende Variablen für Sie angelegt worden:

```
char fAusgabe1='A';
char fAusgabe2= 66;
int  iAusgabe3= 66;
```

Tabelle 15.1

Dezimal	ASCII
64	@
65	A
66	B
67	C

Kreuzen Sie mit Hilfe der Tabelle 15.1 die richtige Ausgabe der folgenden printf-Ausdrücke an (Single Choice):

```
printf("%c %c %i", fAusgabe1, fAusgabe2, iAusgabe3);
```

☒ A B 66 ☐ A 66 66
☐ 65 66 B ☐ keine Ausgabe

```
printf("%c %i %i", fAusgabe1, fAusgabe2, iAusgabe3);
```

☐ A B 66 ☒ A 66 66
☐ 65 66 B ☐ keine Ausgabe

b) Wandeln Sie das folgende Aktivitätsdiagramm (Abb. 15.1) in C-Code um, indem Sie das gegebene Code-Gerüst der out-Funktion vervollständigen:

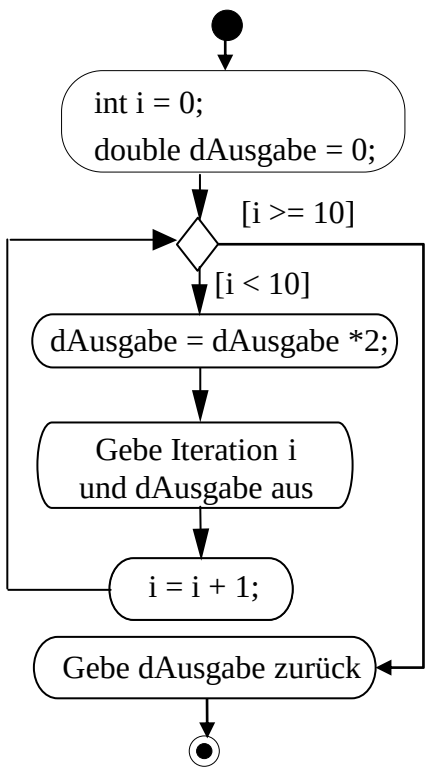


Abb. 15.1

```
double out(){

 int i = 0;                          Alt. zu
 double dAusgabe = 0;                i<10: i<=9

 for(   int   i = 0; i<10; i++   )
 {
    dAusgabe = dAusgabe * 2;

   printf(„Iteration: %i“,     i      );

   printf(„Ausgabe   %f “,   dAusgabe  );
                        Alt.: %lf
 } // Ende for-Schleife

  return dAusgabe;

}// Ende out-Funktion
```

Aufgabe 16: Grundlagen in C – Pointer und Arrays

a) Erstellen Sie einen Integer-Array `iFeld` der Größe **fünf** und weisen Sie dem Array aufsteigend die Werte 1 bis inklusive 5 zu. Verzichten Sie dabei auf Schleifen.

```
int iFeld[5] = {1, 2, 3, 4, 5};
```

b) Nach der Erstellung des Arrays `iFeld` sollen Sie nun einen Integer Pointer `pA` deklarieren und auf das erste Feld vom Array `iFeld` initialisieren:

```
int *pA = iFeld;
```

Alternativ:

```
int *pA;  pA = &iFeld[0];
```

Speichern Sie mit dem zuvor initialisierten Pointer `pA` den Wert der dritten Position des Arrays `iFeld` in die Variable `int iErgebnis`:

```
int iErgebnis = 0;
```

```
iErgebnis = *(pA + 2);
```

c) Vervollständigen Sie den unten aufgeführten `printf`-Befehl, indem Sie die Adresse und den zugehörigen Wert der fünften Stelle des Arrays `iFeld` mit dem Poniter `pA` ausgeben lassen. Gehen Sie davon aus, dass der Pointer `pA` auf die erste Stelle des Arrays `iFeld` zeigt (siehe Tabelle 16.1). Für die Konsolenausgabe ist der dem Array `iFeld` zugehörige Speicher in Tabelle 16.1 gegeben:

Datenspeicher	
Adresse	Wert
0x0000	-1
0x0004	1
0x0008	2
0x000C	3
0x0010	4
0x0014	5
0x0018	0
0x001C	0
0x0020	0
0x0024	0
0x0028	0
0x002C	0

Hinweis: %p soll den hexadezimalen Wert der Adresse, auf die ein Pointer zeigt, angeben.

```
printf("Adresse %p und Wert %i",
        (pA + 4)   ,   *(pA + 4)   );
```

Welche Ausgabe erscheint in der Konsole?

Adresse 0x0014 und Wert 5

Tabelle 16.1 Veranschaulichung der **Speicheradresse** und **4 Bytes Integer Werten** mit dem Pointer `pA` auf der ersten Stelle von Array `iFeld`.

Aufgabe 17: Ergänzen Sie das UML-Zustandsdiagramm

Das bisher betrachtete **Elektrofahrzeug** (E-Auto) verfügt über ein automatisiertes Assistenzsystem (Abb. 17.1). Die Zustände des vereinfachten Assistenzsystems sollen im Folgenden betrachtet und modelliert werden, siehe Abbildung 17.2.

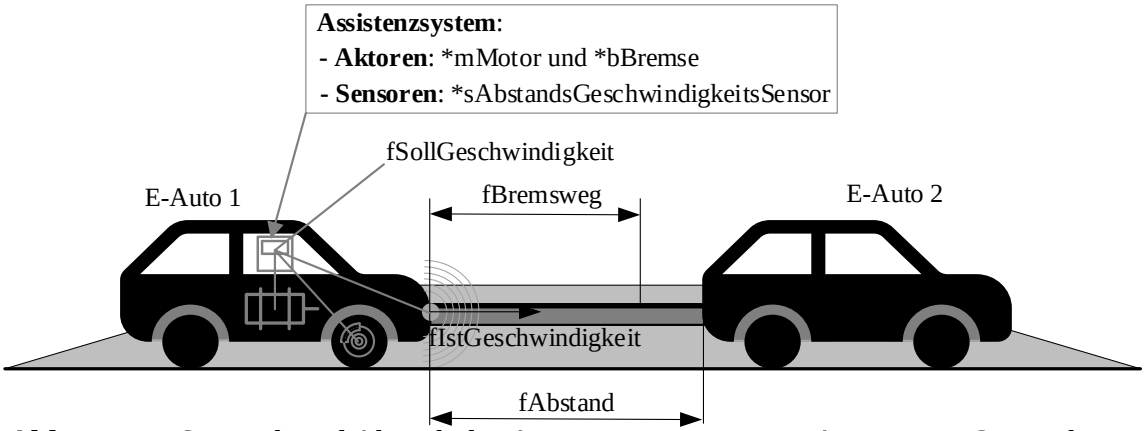


Abb. 17.1 – Sensorik und Aktorik das Assistenzsystems im E-Auto 1. Die Sensorik wird ausschließlich durch die Funktion *AusweichmanoeverPruefen*() aufgerufen.

Wenn das E-Auto 1 im **Zustand „*Anfahrend*“** ist, wird einmalig eingangs der Motor gestartet (**mMotor.start()*). Anschließend wird über die Funktion *SicherAnfahren()* im Zustand wiederholt geprüft, ob das Auto 1 sicher anfahren kann. Beim Verlassen des Zustandes wird die Soll-Geschwindigkeit (*fSollGeschwindigkeit*) gleich *50 km/h* gesetzt und die Bremse (**mBremse.stop()*) gelöst. Sobald die Soll-Geschwindigkeit der Ist-Geschwindigkeit (*fIstGeschwindigkeit*) entspricht, geht das E-Auto 1 in den **Zustand „*Fahrend*“** über. Im Zustand Fahrend, wird kontinuierlich geprüft, ob das Auto am Ziel (*ZielErreicht()*) ist oder ob ein Ausweichmanöver (*AusweichmanoeverPruefen()*) eingeleitet werden muss.

Wenn die Funktion *AusweichmanoeverPruefen()* im **Zustand „*Fahrend*“** den Wert *True* zurückgibt, soll beim Verlassen des Zustandes „Fahrend“ der Bremsweg (*BremswegBerechnen()*) berechnet werden. Im Fall, dass der Bremsweg (*fBremsweg*) kleiner gleich dem Abstand (*fAbstand*) zu E-Auto 2 ist, soll das E-Auto 1 in den **Zustand „Beschleunigend“** übergehen. Eingehend in den Zustand Beschleunigend wird die Soll-Geschwindigkeit (*fSollGeschwindigkeit*) einmalig verdoppelt. Während dieses Zustandes soll wiederholt geprüft werden, ob ein Ausweichen noch immer nötig ist (*AusweichmanoeverPruefen()*). Wenn kein weiteres Ausweichen nötig ist, soll das E-Auto 1 wieder in den Zustand „Fahrend“ übergehen.

Wenn im Zustand „Fahrend“ die Funktion *AusweichmanoeverPruefen()* den Wert *True* zurückgibt, aber der Bremsweg (*fAbstand*) größer als der Abstand (*fAbstand*) zum E-Auto 2 ist, soll das E-Auto 1 in den **Zustand „Bremsend“** übergehen. Nach dem Bremsen befindet sich das Auto im **Zustand „Stehend“** und geht dann wieder in den Zustand „Anfahrend“ über.

Sobald im Zustand „Fahrend“ die Funktion *ZielErreicht()* den Wert *True* zurück gibt, ist das E-Auto 1 am Ziel und soll in den **Zustand „Ankommend“** übergehen. Wenn die Ist-Geschwindigkeit der Soll-Geschwindigkeit entspricht, wird in den **Endzustand** übergegangen.

Füllen Sie die durch römische Ziffern gekennzeichneten Lücken aus dem Zustandsdiagramm (Abb. 17.2) im folgenden Lösungsfeld aus, bzw. beantworten Sie die gegebenen Fragen mit Hilfe der Beschreibung auf der vorherigen Seite:

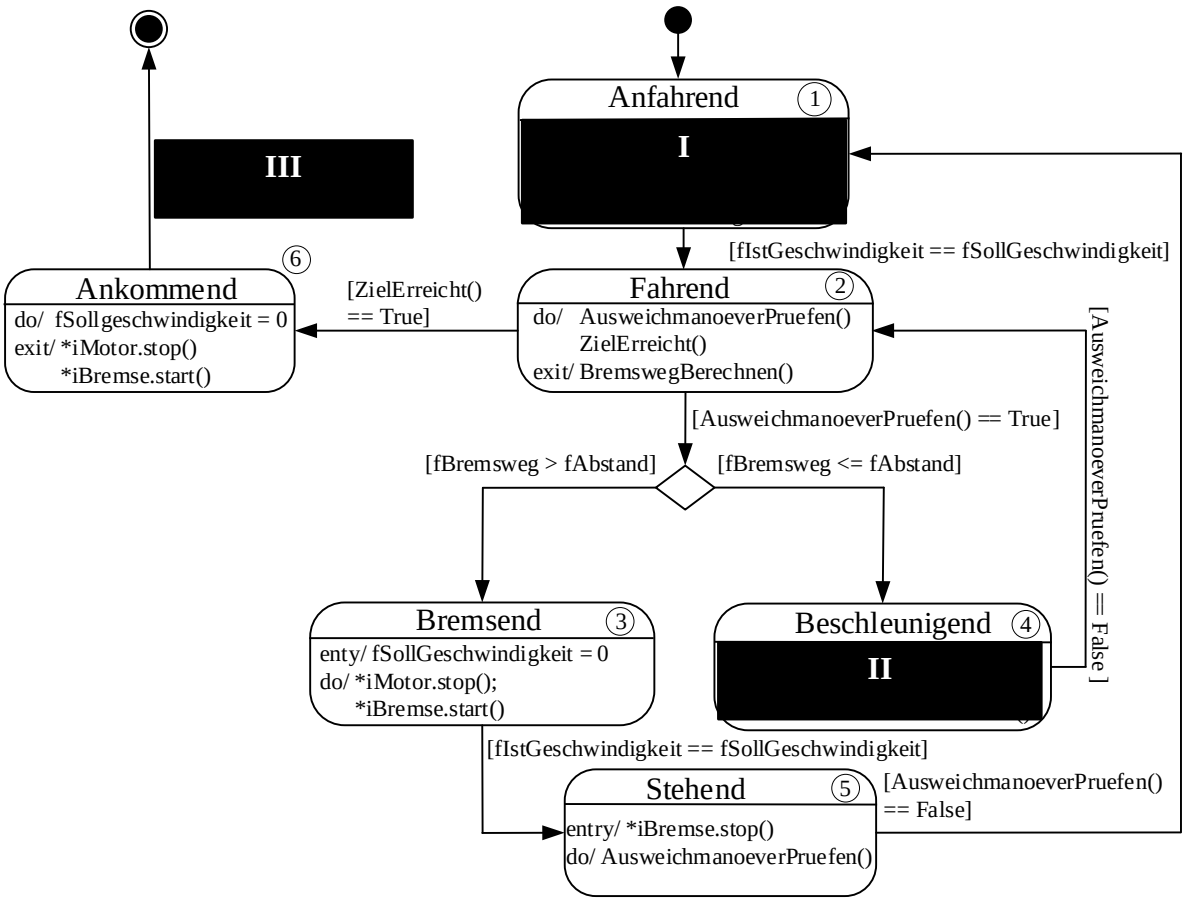


Abb. 17.2 – Zustandsdiagramm für das Assistenzsystem des E-Autos 1.

I. Modellieren Sie den Zustand 1 „`Anfahrend`“ korrekt aus:

entry/ *mMotorStart()

do/ SicherAnfahren()

exit / *iBremse.stop();

fSollGeschwindigkeit = 50

II. Modellieren Sie den Zustand 4 „`Beschleunigend`“ korrekt aus :

entry/ fSollGeschwindigkeit = 100 (km/h)

alternativ …= 2*fSollGeschwindigkeit;

do/ AusweichmanoeverPruefen()

III. Geben Sie die Übergangsbedingung in den Endzustand an:

[fIstGeschwindigkeit == fSollGeschwindigkeit]

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 implementierten Funktionen des Systems.

Typ	Name	Beschreibung
VARIABLEN	int iZustand	Integer, der den Zustand des Systems angibt
	MOTOR *mMotor	Pointer für den Motortreiber-Baustein von Motor von E-Auto 1
	BREMSE *bBremse	Pointer für den Bremsentreiber-Baustein von E-Auto 1
	float fBremsweg	Aktueller Bremsweg von E-Auto 1
FUNKTIONEN	void initsystem(MOTOR *m_Motor, BREMSE * b_Bremse,)	Initialisierungsfunktion des Systems: Initialisierung der Pointer für die Sensorik und Aktorik.
	bool AusweichmanoeverPruefen()	Prüft, ob ein Manöver nötig ist
	bool ZielErreicht()	Gibt an, ob das E-Auto 1 am Ziel ist
	float GeschwindigkeitMessen()	Bestimmt die aktuelle Ist-Geschwindigkeit von E-Auto 1.
	float BremswegBerechnen()	Bestimmt den aktuellen Bremsweg von E-Auto 1.

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

Zusätzlich zu den in **Tabelle 18.1** angegebenen Variablennamen sind die gegebenen Funktionen bereits im Header `assistenzsystem.h` des Assistenzsystems definiert und implementiert. Im weiteren Verlauf ist der Übergang von Zustand 2 zu den Zuständen 3, 4 und 6 wie folgt (Abbildung 18.1) gegeben:

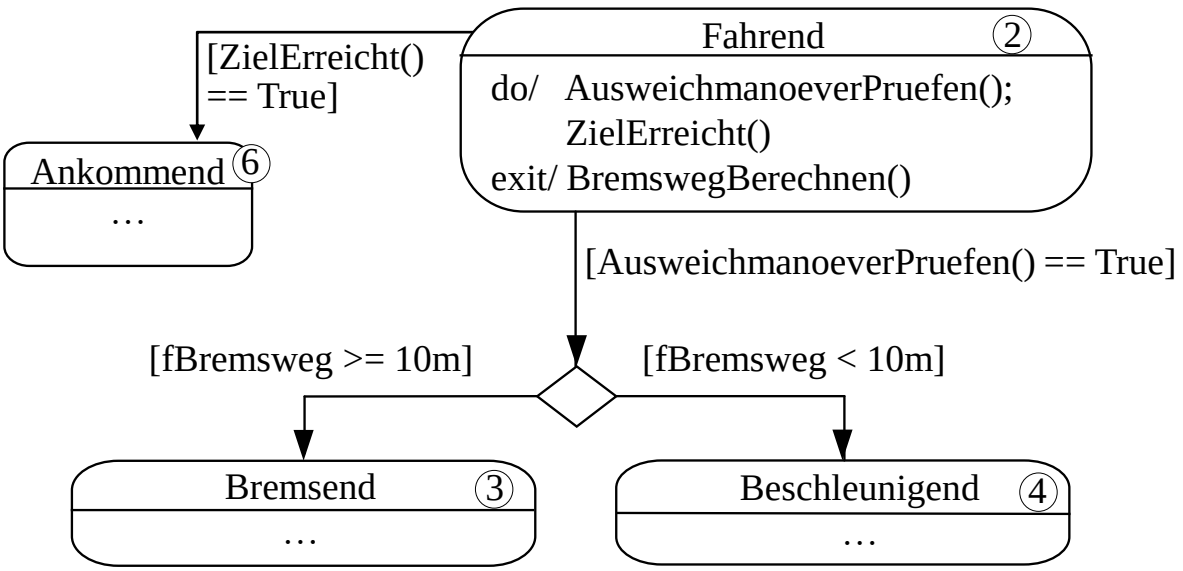


Abbildung 18.1

Vervollständigen Sie das im folgenden Lösungskästchen gezeigte Programmgerüst gemäß der Kommentare und den in **Abbildung 18.1** und **Tabelle 18.1** gegebenen Informationen in der Programmiersprache C.

```
//Header des Assistenzsystems einbinden
  #include "assistenzsystem.h", alternativ <>
_____________________________________

// Initialisierung mit Variablen und der Funktion aus Tabelle

MOTOR *mMotor;

BREMSE *bBremse;
   initSystem(mMotor, bBremse)
_____________________________________________________________;

// Setzen Sie die Zustandsvariable iZustand auf Zustand 2
   iZustand = 2;
_____________________________________

int main() {
 // Zyklische Endlosausfuehrung
   while(True)  - alternativ while(1) …
_____________
 { // Fuellen Sie den Zustandsautomat aus
       switch ( iZustand)
_____________________________________
       {
            case 1:
                 /* Zustand 1 */
            case 2:
                 if (ZielErreicht()    // ZielErreicht() == True
_____________________________){
                    GeschwindigkeitMessen();
                     BremswegBerechnen()
                    _____________________________;
                     iZustand = 6
                    _____________________;
                 }
                 if ( AusweichmanoeverPruefen() == True
_____________){
                    GeschwindigkeitMessen();
                    BremswegBerechnen();
                    if ( fBremsweg >= 10
_____________________) {
                         iZustand=3
                        _____________________________; }
                    if ( fBremsweg < 10
_____________________) {
                         iZustand=4
                        _____________________________;}
                 }
                         break
                 _____________________; // Zustand 2 verlassen
      /* weitere ZUSTAENDE*/
       }
 }
return 0;}
```

Alternativ: andere Reihenfolge der Bedingungen

Aufgabe 19: Algorithmen

Ein Elektroauto verfügt über acht Parksensoren (siehe **Abb. 19.1),** welche die Abstände zu anderen parkenden Autos messen können. Zur Ansteuerung der Sensoren stehen Ihnen folgende Funktionen zu Verfügung:

- `float measuresens(SENSOR *Sensor1)`: Führt eine Messung mit Sensor1 aus und gibt den Abstand zurück.
- `void adjustsens(SENSOR *Sensor1)`: Stellt den Sensor1 nach einer fehlerhaften Messung neu ein.

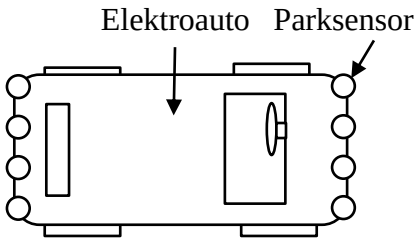


Abb. 19.1 – Skizze: Elektroauto mit Parksensoren

Die Funktion `sensorcal`, die einen Sensor kalibriert, ist wie folgt zu implementieren:

- Der Abstand soll durch den Sensor (`SENSOR *Sensor1`) gemessen (`measuresens`) und der Messwert in der Variablen `dist` abgespeichert werden.
- Ist der gemessene Abstand kleiner als 100, muss der Sensor neu eingestellt werden. Das Einstellen des Sensors erfolgt mit der Funktion `adjustsens`. Ist der Abstand des Sensors größer gleich 100 war die Kalibrierung des Sensors erfolgreich und die Funktion `sensorcal` kann beendet werden.
- Nach fünf Versuchen, den Sensor einzustellen, wird die Funktion `sensorcal` abgebrochen. Nutzen Sie dafür die gegebene While-Schleife und die Variable `counter`.

Hinweise:

- *Die Anzahl an Linien im Lösungskästchen ist bei allen folgenden Aufgaben unabhängig von der Anzahl an geforderten Code-Linien.*
- *Es müssen keine Header eingebunden werden.*

a) Implementieren Sie die Funktion `sensorcal` in der Programmiersprache C gemäß der obigen Beschreibung.

```
void sensorcal(SENSOR *Sensor1)
{ float dist = 0.0;
  int counter = 0;
  while(1)
  {   dist = measuresens(Sensor1);
      Alt.: if(counter>5){break;}
       if (dist < 100.0){
         adjustsens(Sensor1);
       }
       else{
         break; alternativ return;
       }

       if (counter >= 5){
          break; alternativ return;
       }
       counter ++;
  }
}
```

Aufgabe 19: Algorithmen

Die gemessenen Abstände der acht Sensoren sollen im Folgenden durch die Funktion `filterminmax` aufbereitet werden. Die Funktion `filterminmax` hat als Übergabeparameter das Array `float sensval[8],` wobei in jedem Arrayfeld ein Messwert abgelegt ist. Die Ergebnisse von `filterminmax` sind in dem Array `float filres[7]` abzuspeichern. Die Funktion `filterminmax` ist wie folgt zu implementieren (siehe **Abb. 19.2**):

- Aus jeweils **zwei** (n=2) aufeinanderfolgenden Messwerten (`sensval`) ist der geometrische Mittelwert **GM** zu bilden (siehe Formel 1).
- Liegt der geometrische Mittelwert **GM** zwischen `minvalue` und `maxvalue`, wird der Wert in `filres` gespeichert
- Ist der geometrische Mittelwert kleiner als `minvalue` oder größer als `maxvalue`, wird der Wert 0.0 in `filres` gespeichert
- Als Rückgabewert der Funktion `filterminmax` ist das Array `filres[7]` auszugeben.

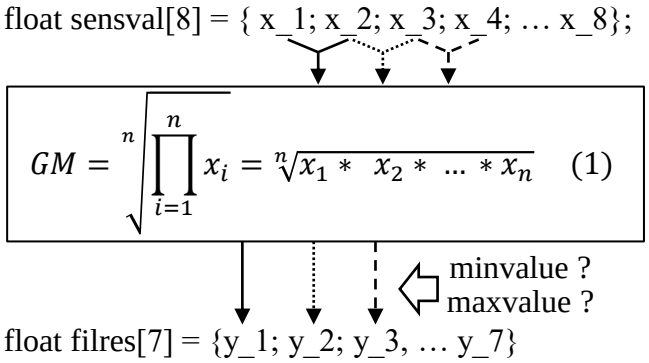


Abb. 19.2 – Skizze: Funktion `filterminmax`

Hinweis: Es steht Ihnen die Funktion float sqrtn(float x, int n) → $\sqrt[n]{x}$ *zur Verfügung.*

b) Implementieren Sie die Funktion `filterminmax` in der Programmiersprache C gemäß der obigen Beschreibung.

```
float *filterminmax(float *sensval, float *filres)
{ float maxvalue = 10.0;
  float minvalue = 0.1;

for (int i=0; i<7; i=i+1)
{
   filres[i] =
       sqrtn(*(sensvalues+i) * *(sensvalues+i+1),2)

    if (filres[i] < minvalue || filres[i] > maxvalue)
    {
        filres[i] = 0.0;
    }
}

return filres;
}
```

Aufgabe 20: Datenstrukturen

Die Sensoren werden im weiteren Verlauf als doppelt verkette Liste betrachtet, siehe **Abb. 20.1** und **Abb. 20.2**.

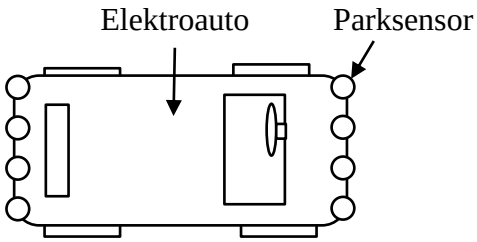


Abb. 20.1 – Skizze: Elektroauto mit Parksensoren

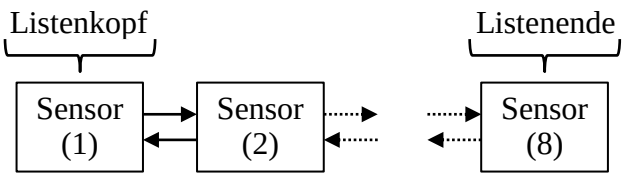


Abb. 20.2 – Skizze: Sensoren als doppelt verkette Liste Sensorliste

In einem Listenelement vom Typ `SENSOR` soll das Folgende gespeichert werden:

- Ein Zeiger (`nextsensor`), welcher auf das nächste Listenelement vom Typ `SENSOR` und somit auf den nächsten Sensor zeigt.
- Ein Zeiger (`prevsensor`), welcher auf das vorherige Listenelement vom Typ `SENSOR` und damit auf den vorherigen Sensor zeigt.
- Die Ausgabevariable (`outputsensor`) als Fließkommazahl mit dem Datentyp Float, in der der Abstand zu einem parkenden Auto gespeichert wird.
- Der Status des Sensors (`status`) mit dem Datentyp Char, in dem fehlerhaftes Verhalten des Sensors gespeichert werden kann.

Hinweise:

- *Für alle folgenden Teilaufgaben können Sie annehmen, dass die verkette Liste SENSOR initialisiert und mit Daten gefüllt wurde.*

a) Vervollständigen Sie das nachfolgende Lösungskästchen zur Definition eines Listenelementes vom Typ `SENSOR` gemäß der obigen Beschreibung unter Verwendung der Programmiersprache C.

```
typedef struct SENSOR ___________{
    struct SENSOR *nextsensor;_____________________
    struct SENSOR *prevsensor;_____________________
    float outputsensor;____________________________
    char status;___________________________________
_______________________________________________
_______________________________________________
_______________________________________________
_______________________________________________
} SENSOR;
```

Aufgabe 20: Datenstrukturen

Der Status von acht Sensoren soll in einer Datei `statuslog.txt` gespeichert werden. Die Funktion `storestatus` hat dabei die Übergabevariable `slisthead`, die auf den Listenkopf und somit auf das erste Listenelement (Sensor (1)) der Sensorliste zeigt. Der Status eines Sensors ist in der Variablen `status` des jeweiligen Listenelements gespeichert und hat entweder den Wert `'E'` für *Error* oder `'W'` für *Working*. In der Datei sollen die Statuswerte der acht Sensoren, mit einem Komma voneinander abgetrennt, gespeichert werden. Die Datei soll dabei nur für den Schreibvorgang geöffnet und danach geschlossen werden.

b) Implementieren Sie die Funktion `storestatus` in der Programmiersprache C gemäß der obigen Beschreibung.

Hinweis:

- *Beispiel für einen Datensatz:*

```
W,E,W,W,E,W,E,W,
```

```
void storestatus(SENSOR *slisthead)
{
 FILE *p = fopen("statuslog.txt", "w");

 for (int i=0; i<8; i++)
 {
   fprintf(p, "%c", slisthead->status);
   slisthead = slisthead->nextsensor;
 }

 fclose(p);
}
```

Aufgabe 20: Datenstrukturen

Zu den bestehenden acht Sensoren soll ein neuer Sensor (9) (`sensornine`) vom Typ `SENSOR` mithilfe der Funktion `addlist` hinzugefügt werden, siehe **Abb. 20.3**. Die Funktion `addlist` hat dabei die Übergabevariable `slisttail`, die auf das Listenende und somit auf das letzte Listenelement (Sensor 8) der Sensorliste zeigt.

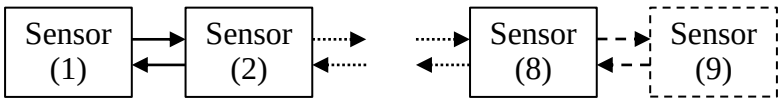


Abb. 20.3 – Skizze: Sensoren als doppelt verkette Liste mit zusätzlichem Sensor (9)

Die Funktion `addlist` ist wie folgt zu implementieren:

- Reservieren Sie Speicher in der Größe eines Listenelements und richten Sie den Pointer `sensornine` darauf aus.
- Das Struct-Element `nextsensor` des Listenelements `sensornine` wird mit `NULL` initialisiert.
- Das Struct-Element `prevsensor` des Listenelements `sensornine` wird mit dem Übergabeparameter `slisttail` initialisiert.
- Das Struct-Element `nextsensor` des Listenelements `slisttail` wird mit dem neuen Sensor `sensornine` initialisiert.
- Die Funktion `addlist` soll als Rückgabewert einen Pointer auf das Listenelement `sensornine` ausgeben.

c) Implementieren Sie die Funktion `addlist` in der Programmiersprache C gemäß der obigen Beschreibung.

```
SENSOR * addlist(SENSOR *slisttail)
{SENSOR *sensornine = NULL; //neuer Sensor
_____________________________________________
_____________________________________________
  sensornine = (SENSOR*) malloc(sizeof(SENSOR));
_____________________________________________
  sensornine->nextsensor = NULL;
_____________________________________________
  sensornine->prevsensor = slisttail;
_____________________________________________
  slisttail->nextsensor = sensornine;
_____________________________________________
  return sensornine;
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
_____________________________________________
}
```