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

Vorname:	
Nachname:	
Matrikelnummer:	

Prüfung – Informationstechnik

Sommersemester 2012

31. August 2012

Bitte legen Sie Ihren Lichtbildausweis bereit.

Sie haben für die Bearbeitung der Klausur 120 Minuten Zeit.

Diese Prüfung enthält **24** nummerierte Seiten inkl. Deckblatt.

Bitte prüfen Sie die Vollständigkeit Ihres Exemplars!

Bitte schreiben Sie nicht mit rot oder grün Farbe! Bitte verwenden Sie keinen Bleistift!

Diesen Teil nicht ausfüllen.

Aufgabe	GL	BS	MSE	C	∑	**Note**
erreichte Punkte						
erzielbare Punkte	48	48	48	96	240	

Vorname, Name

Matrikelnummer

Punkte

Aufgabe GL: Zahlensysteme und logische Schaltungen

Aufgabe GL: 48 Punkte

a) Überführen Sie die unten angegebenen Zahlen in die jeweils anderen Zahlensysteme. *Wichtig:* Achten Sie genau auf die jeweils angegebene *Basis*!

1 (12)$_7$ = (**9**)$_{10}$ = (**1001**)$_2$

2 (24,875)$_{10}$ = (**11000,111**)$_2$

b) Stellen Sie die Gleitkommazahl 0,35 nach den Prinzipien der IEEE 754 Schreibweise dar (biased Exponent: 3 Bits, Mantisse: 7 Bits). Tragen Sie Ihre Ergebnisse in die dafür vorgesehenen Textblöcke ein.
Wichtig: Ergebnisse und Nebenrechnungen außerhalb der dafür vorgesehenen Textblöcke werden nicht bewertet.

(0,35)$_{10}$ =(0,010110011)$_2$

M = 0110011

E = -2

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

e = $E + B = -2 + 3 = 1$

$Z = (-1)^V * M * 2^E$

1. Schritt: Umrechnen: 0,010110011
2. Schritt: Mantisse *M*=1,0110011 (E=-2)
3. Schritt: Bias berechnen : $B = 2^{(3-1)} - 1 = 3$
4. Schritt: Exponent E berechnen:
a) $e = E + B = -2 + 3 = 1$
b) $(1)_{10} = (001)_2$
4. Schritt: Einsetzen:
$Z = (-1)^0 * (1,0110011)_2 * 2^{-2}$

V	biased Exponent e (3 Bits)	Mantisse (7 Bits)
0	001	0110011

Vorname, Name

Matrikelnummer

Punkte

Eine Uhr besteht aus mehreren 7-Segmentanzeigen, wie in Abbildung 1 dargestellt. Für die Minutenanzeige sollen Sie die linke 7-Segmentanzeige realisieren (die 10-Minutenschritte). Die Ansteuerung wird mit einem Logikbaustein realisiert (Abbildung 2). Für den Logikbaustein „BCD-Code“ der 7-Segmentanzeige erstellen Sie die Logik. In der unten angegebenen Wahrheitstabelle (Tabelle 1) ist die Ansteuerung des Segments „g“ der 7-Segmentanzeige dargestellt, wobei lediglich die Dezimalzahlen 0 bis 5 in der angezeigt werden sollen. Die binären Eingänge x_1, x_2 und x_3 kodieren die Dezimalzahl entsprechend angegebenen Wahrheitstabelle.

Abbildung 1: Skizze einer Uhr mit 7-Segmentanzeigen

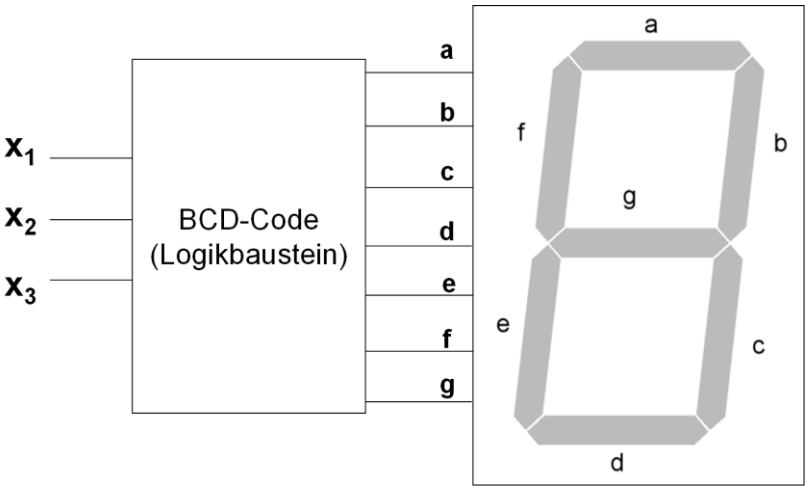


Abbildung 2: Ansteuerung der Segmente mit dem Logikbaustein

Dez	x_1	x_2	x_3	g
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	X
7	1	1	1	X

Tabelle 1: Wahrheitstabelle für das Segment „g“

Vorname, Name

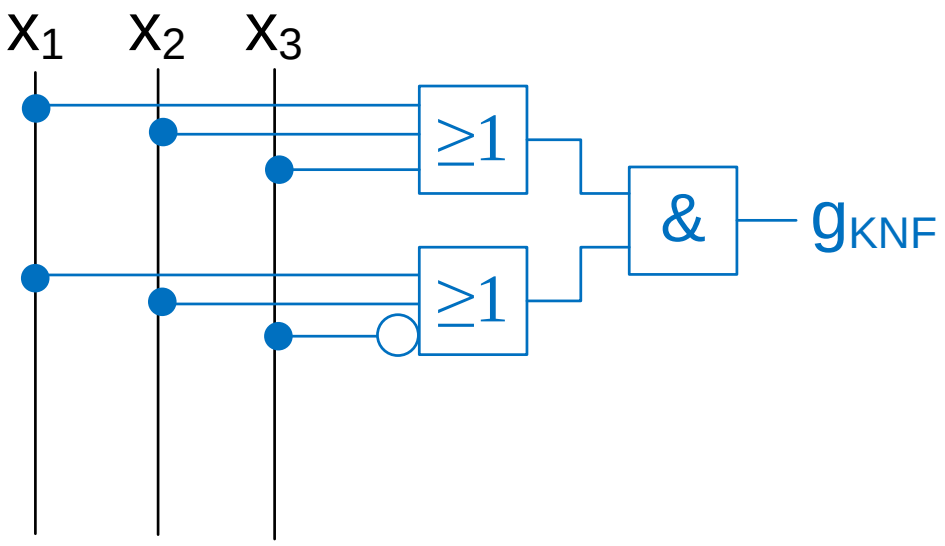
Matrikelnummer

Punkte

c) Erstellen Sie die KNF *(Konjunktive Normalform)* aus der Wahrheitstabelle

$$g_{KNF} = \left(x_1 \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee x_2 \vee \overline{x_3}\right)$$

d) Zeichnen Sie die das *Blockschaltbild* der *KNF*.



e) Minimieren Sie mit Hilfe des *KV-Diagramms* die Ausgangsfunktion für g_{min} in *DNF (Diskunktive Normalform)* und schreiben Sie die minimierte Form in boolescher Algebra auf.

	x_1		$\overline{x_1}$	
x_2	X	X	1	1
$\overline{x_2}$	1	1	0	0
	$\overline{x_3}$		x_3	$\overline{x_3}$

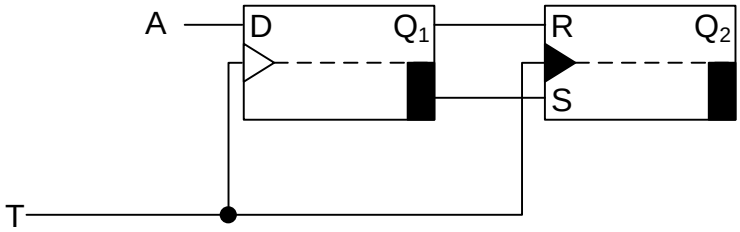
$$g_{min} = \overline{x_2} \vee x_1$$

Vorname, Name

Matrikelnummer

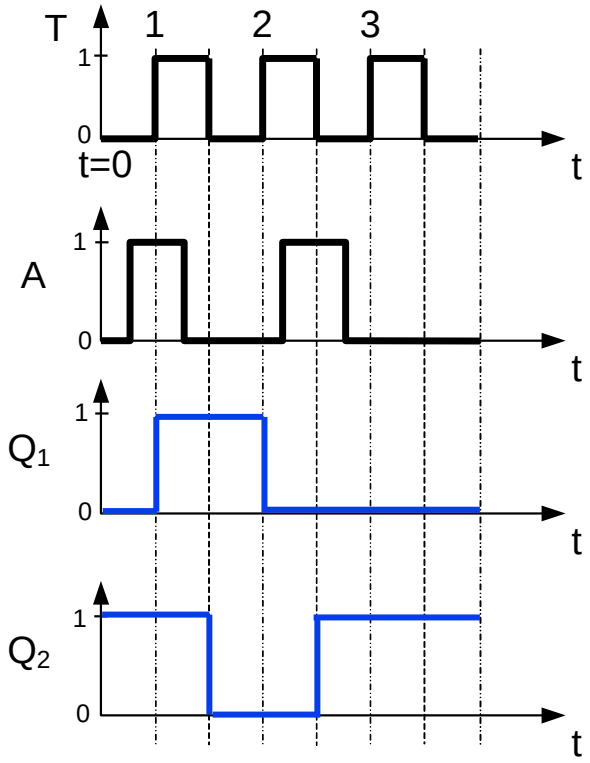
Punkte

Gegeben sei die folgende Master-Slave FlipFlop-Schaltung



Bei t = 0 seien alle Flip-Flops im folgenden Zustand: $Q_1 = 0$ und $Q_2 = 1$

f) Analysieren Sie die Schaltung indem Sie die zeitlichen Verläufe für Q_1 und Q_2 in die vorgegebenen Koordinatensysteme eintragen. Die Signallaufzeiten können dabei vernachlässigt werden. Beachten Sie das die Eingänge am RS-FlipFlop nur der Übersichtlichkeit halber vertauscht sind (Funktionalität ist wie bei einem normalen RS-FlipFlop).



Vorname, Name

Matrikelnummer

Punkte

Aufgabe BS: Betriebssysteme

Aufgabe BS: 48 Punkte

1. Scheduling

Vier Prozesse (P1 bis P4) sollen mit einem Einkernprozessor abgearbeitet werden. Die Abbildung „Zeitverlauf der Prozesse“ gibt die Ausführungszeiten, die Zeiten, zu denen die Prozesse am Einkernprozessor eintreffen, und die Deadlines der einzelnen Prozesse an.
Die vier Prozesse sollen zur Laufzeit mit unterschiedlichen Scheduling-Verfahren geplant werden. Für die Scheduling-Verfahren, bei denen Prioritäten berücksichtigen werden müssen, ist in der Tabelle „Prioritätenverteilung“ die entsprechende Prioritätenverteilung gegeben.

Prozess	Priorität
P1	2
P2	4
P3	3
P4	1

Tab.: Prioritätenverteilung (Prioritätslevel: „1“ = hoch, „4“ = niedrig)

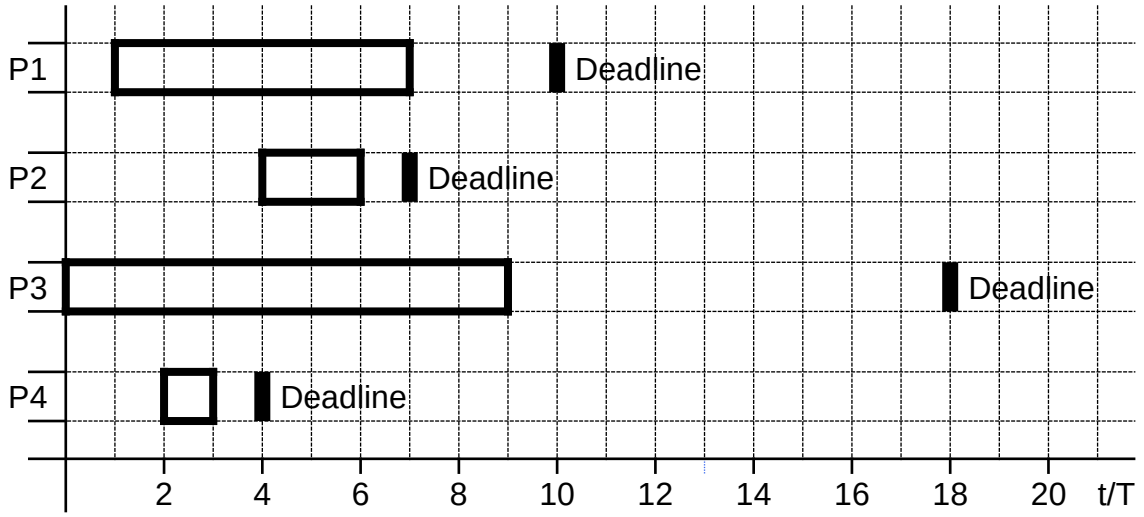


Abb.: Zeitverlauf der Prozesse

Hinweis: Bitte füllen Sie die nachfolgenden Aufgaben nach dem folgenden Schema aus:

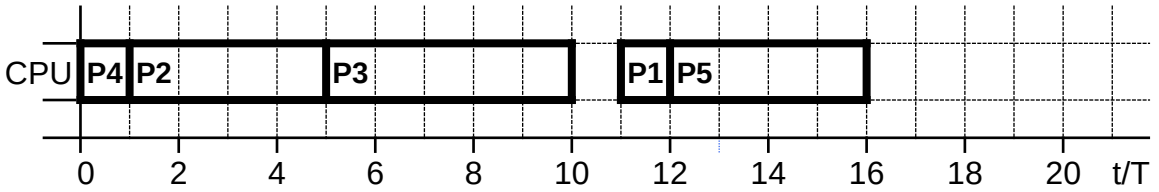


Diagramm: Lösungsbeispiel

Vorname, Name

Matrikelnummer

Punkte

a) In der ersten Teilaufgabe erfolgt die Abarbeitung der oben gegebenen vier Prozesse über die Prioritätenverteilung und mittels eines präemptiven Schedulings.

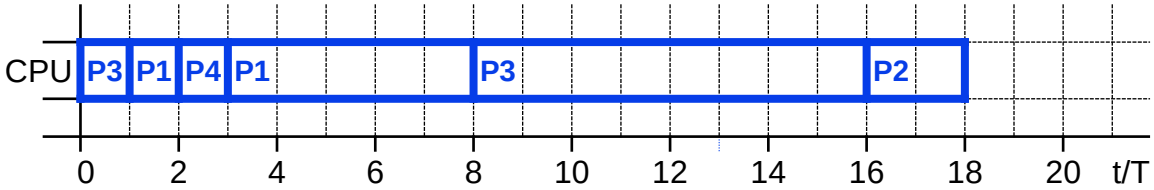


Diagramm: feste Prioritäten

b) In der zweiten Teilaufgabe sollen die vier Prozesse nach dem First In First Out-Verfahren (FIFO) eingeplant werden.

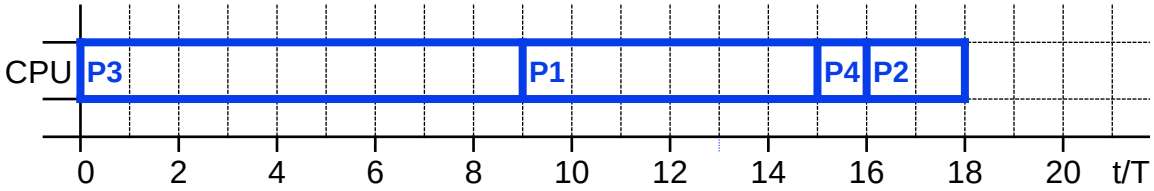


Diagramm: First In First Out (FIFO)

c) In der dritten Teilaufgabe sollen die vier Prozesse nach dem Earliest Deadline First-Verfahren (EDF) abgearbeitet werden. Das Scheduling erfolgt präemptiv.

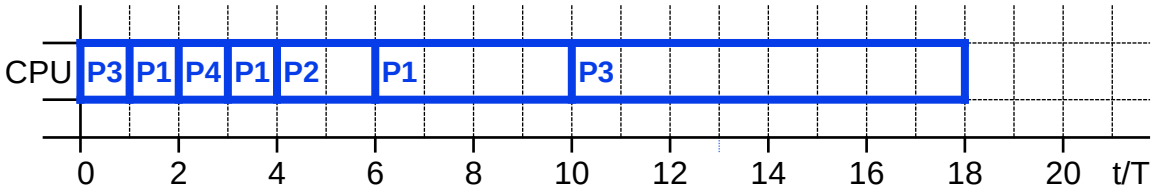


Diagramm: Earliest Deadline First (EDF)

d) Welche der letzten beiden Schedulingverfahren (FIFO, EDF) würde sich für ein Echtzeitbetriebssystem eignen? Begründen Sie Ihre Entscheidung.

EDF, weil die Deadlines der Prozesse berücksichtigt bzw. eingehalten werden.

Vorname, Name

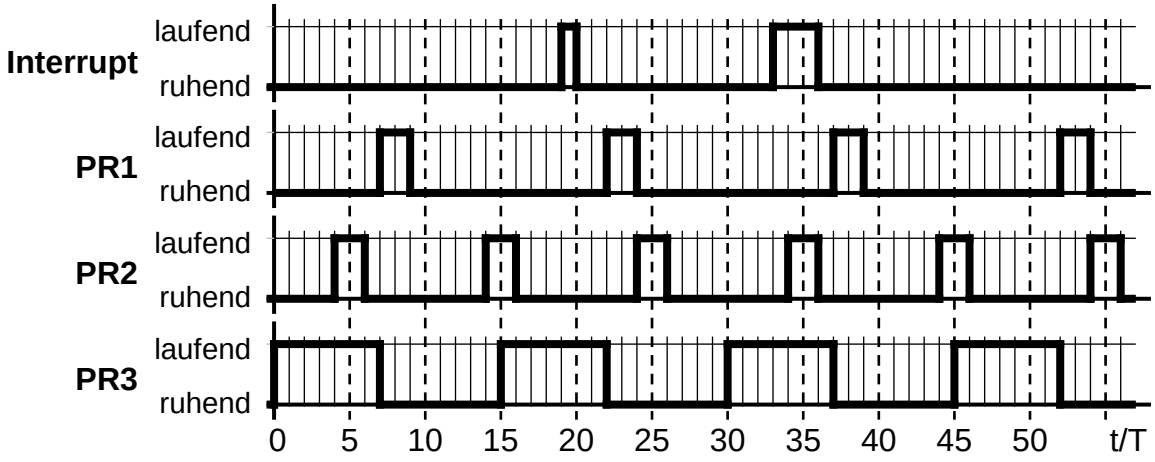
Matrikelnummer

Punkte

2. Asynchrone Programmierung

Drei periodische und präemptive Prozesse (PR1 bis PR3) sollen mit dem Verfahren der asynchronen Programmierung auf einem Einkernprozessor ausgeführt werden. Der Prozess PR1 besitzt die höchste, der Prozess PR3 die niedrigsten Priorität. Die Ausführung wird zweimal durch einen Interrupt unterbrochen.

Tragen Sie in das unten angegebene Diagramm die tatsächliche Abarbeitung der Rechenprozesse nach dem Verfahren der asynchronen Programmierung ein.



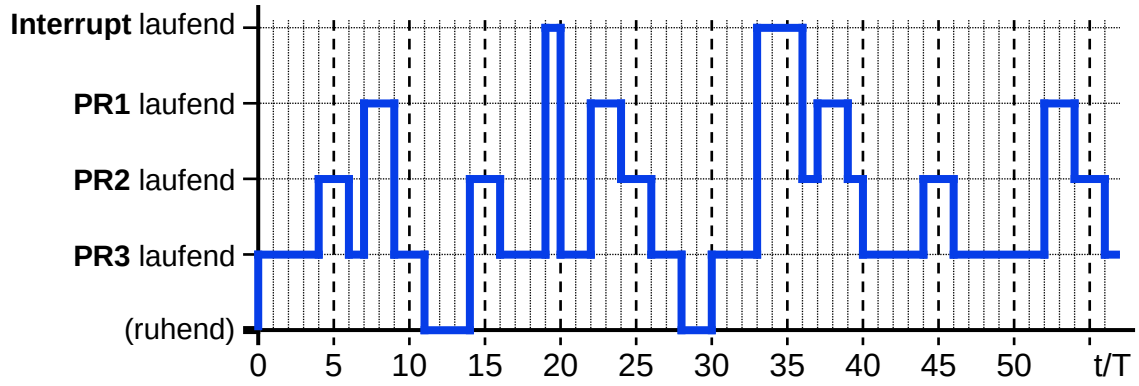


Diagramm: asynchrone Programmierung

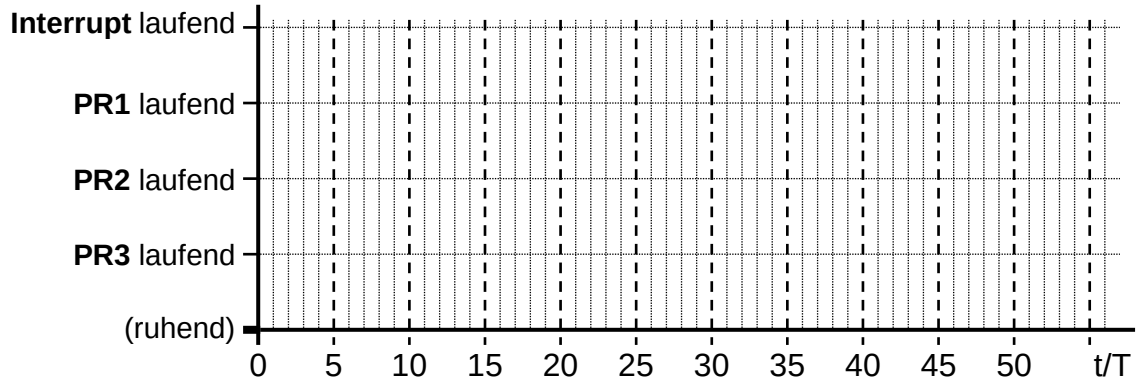


Diagramm: asynchrone Programmierung – Ersatzfeld, falls oberes Feld falsch

Vorname, Name

Matrikelnummer

Punkte

3. Semaphoren

Gegeben ist die Anordnung von Semaphor-Operationen am Anfang und am Ende der Tasks A,B,C. Ermitteln Sie für die Fälle I und II, ob und in welcher Reihenfolge diese Tasks bei der angegebenen Initialisierung der Semaphor-Variablen ablaufen. Geben Sie zusätzlich an, ob sich die Taskreihenfolge wiederholt bzw. um welche Art von Verklemmung es sich handelt.

Hinweis: Die für die Ausführung eines Tasks notwendigen Semaphoren sollen nur im Block verwendet werden (z.B. Task A startet nur, wenn alle Semaphoren S2, S2, S3, S3 frei sind). Sind mehrere Tasks ablauffähig, gelten folgende Prioritäten: A: hoch, B: mittel, C: niedrig. P(Si) senkt Si um 1, V(Si) erhöht Si um 1. Geben Sie die Reihenfolge der ablaufenden Tasks in folgender Schreibweise an: z.B. ABC$\overline{\text{ABB}}$.

Tasks

Taskabarbeitung ↓

A	B	C
P(S2)		
P(S2)	P(S1)	P(S3)
P(S3)	P(S3)	
P(S3)		
…	…	…
V(S3)	V(S2)	V(S1)
V(S3)	V(S3)	

Fall I:

Task	Semaphoren S1	S2	S3
Start	1	1	2
B	0	2	2
A	0	0	2
C	1	0	2
B	0	1	2
C	1	1	2

Fall I: $\overline{BACBC}$

→ **Wiederholung**

Fall II:

Task	Semaphoren S1	S2	S3
Start	2	0	1
B	1	1	1
B	0	2	1
C	1	2	1
B	0	3	1
C	1	3	1
B	0	4	1
C	1	4	1

Fall II: $BBC\overline{BC}$

→ **partieller Deadlock/Livelock**

Vorname, Name

Matrikelnummer

Punkte

Aufgabe MSE: Modellierung und Software Entwicklung

Aufgabe MSE: 48 Punkte

Die folgende Beschreibung eines Lagers für Material benötigen Sie zur Bearbeitung der Teilaufgaben SysML a) und b) und SA/RT a) und b):

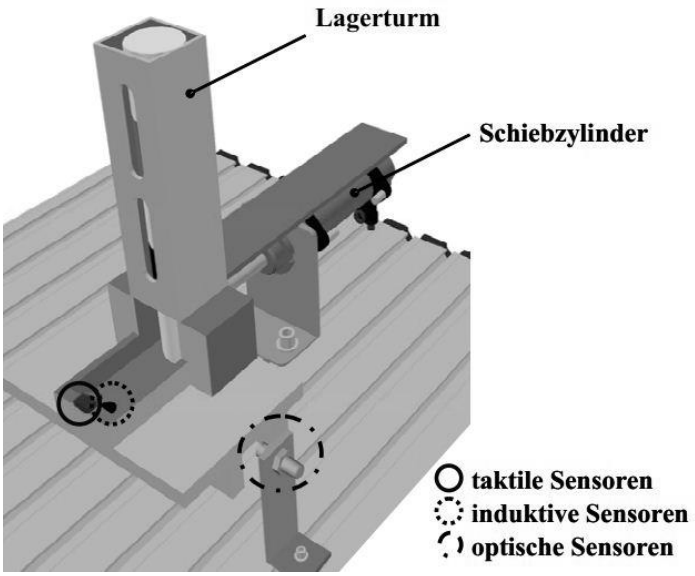


Tabelle Materialerkennungslogik:

Messwert Sens_induktiv	Messwert Sens_optisch	Material
0	0	Schwarzer Kunststoff
0	1	Weißer Kunststoff
1	1	Aluminium

1. SysML

Aufbau des Lagers (Wird für Aufgabenteil SysML a) und b) benötigt):
Das Lager kann *angeschaltet* und *ausgeschaltet* werden. Es besteht aus einem Schiebzylinder und drei Sensoren. Der Schiebzylinder hat zwei Werte, um die Endlagen des Kolbens anzugeben (*vorne* (1 == vorne, 0 == nicht vorne); *hinten* (1 == hinten, 0 == nicht hinten)). Außerdem ist er in der Lage *nach vorne zu fahren* und *zurück zu fahren*. Die drei Sensoren (*Sens_taktil*, *Sens_optisch* und *Sens_induktiv*) sind alle binäre Sensoren (*bool*) und liefern jeweils einen Messwert.

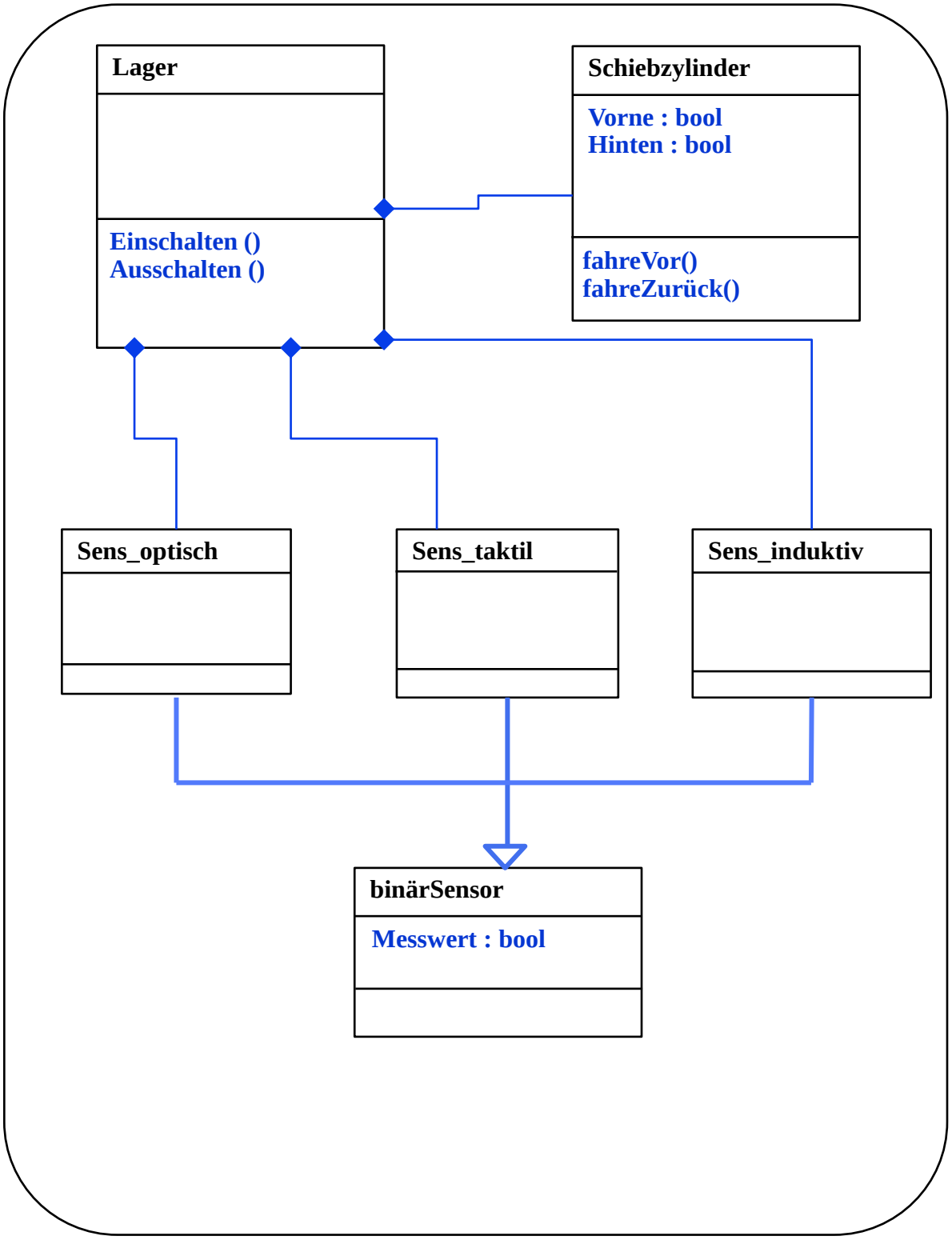
Ablauf des Ausstoßens (Wird für Aufgabenteil SysML b) benötigt):
Zu Beginn muss das Lager einmal *eingeschaltet* werden. Ist die vordere Position nicht besetzt (*Sens_taktil* == 0) wird ein Werkstück vereinzelt. Dafür wird der Kolben des Schiebzylinders nach *vorne gefahren* und anschließend wieder nach *hinten gefahren*. Es wird sichergestellt, dass beide Aktionen vollständig ausgeführt werden (siehe Endlagen des Kolbens oben). Ist nach diesem Vorgang kein Material vorhanden, wird die Anlage *ausgeschaltet*. Ist jedoch Material vorhanden, wird die Materialerkennung gestartet. Die Materialerkennung erfolgt durch die Sensoren *Sens_induktiv* und *Sens_optisch* und ergibt sich wie in der Tabelle *Materialerkennungslogik* dargestellt. Wurde das Material erkannt, verharrt das Lager im Zustand *abholbereit*, bis die Position vorne wieder frei ist. Ist die Position vorne frei, so beginnt der gesamte Ablauf wieder von vorne (bei bereits eingeschaltetem Lager).

Vorname, Name

Matrikelnummer

Punkte

a) Ergänzen Sie das *gegebene Blockdefinitionsdiagramm.* Bilden Sie dazu die Struktur der Anlage durch Einzeichnen von den entsprechenden Beziehungstypen ab (es ist möglich das ein Block mehrere Beziehungstypen hat). Ergänzen Sie die vorhandenen Blöcke um nötige Attribute, Methoden sowie deren Datentypen. Definieren Sie keine weiteren Blöcke.

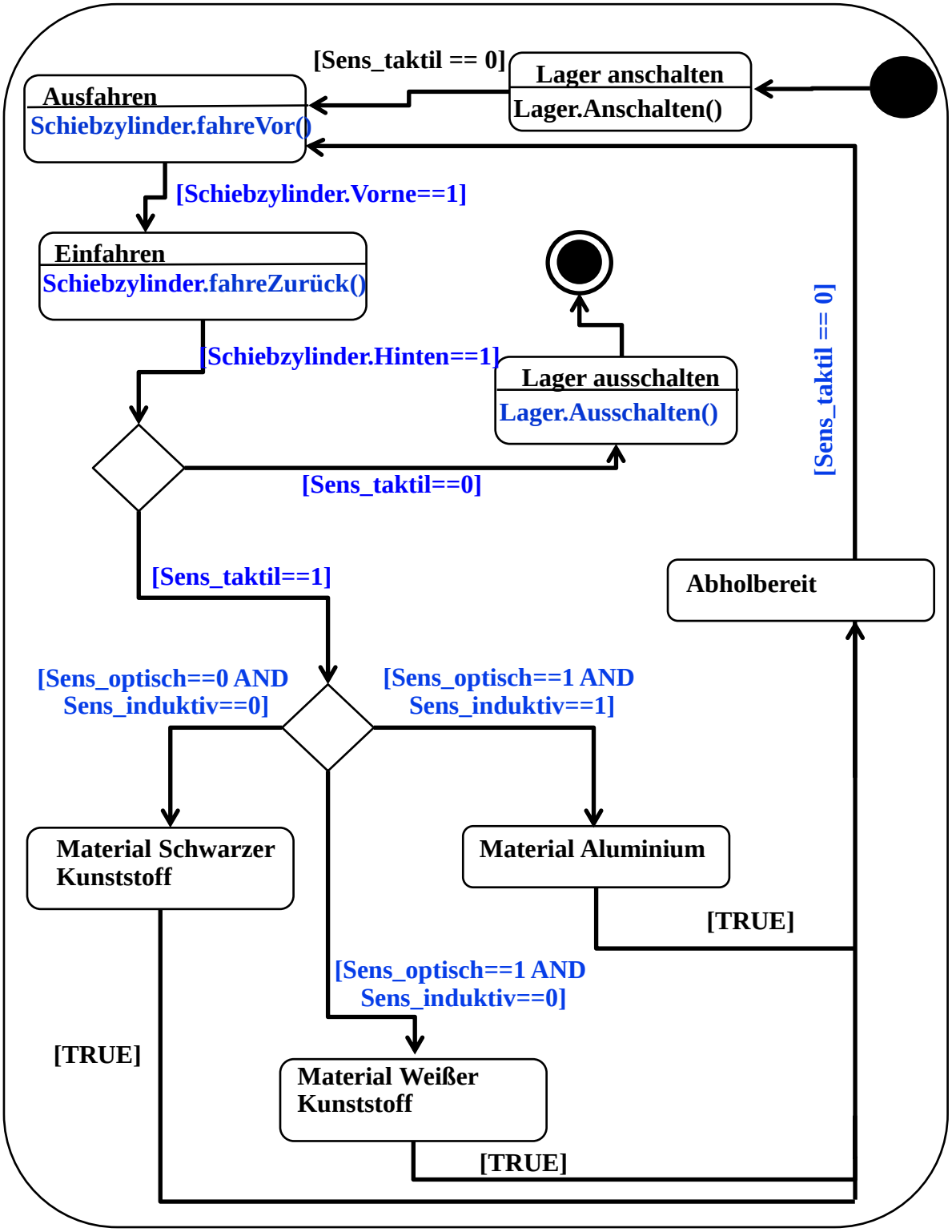


Vorname, Name

Matrikelnummer

Punkte

b) Ergänzen Sie das *gegebene Zustandsdiagramm*, um *alle beschriebenen Funktionen* der Anlage gemäß dem Aufgabentext abzubilden. Bilden Sie das Verhalten der Anlage mit den gegebenen Zuständen ab und ergänzen Sie nötige Übergangsbedingungen und Methodenaufrufe. Definieren Sie keine neuen Zustände.



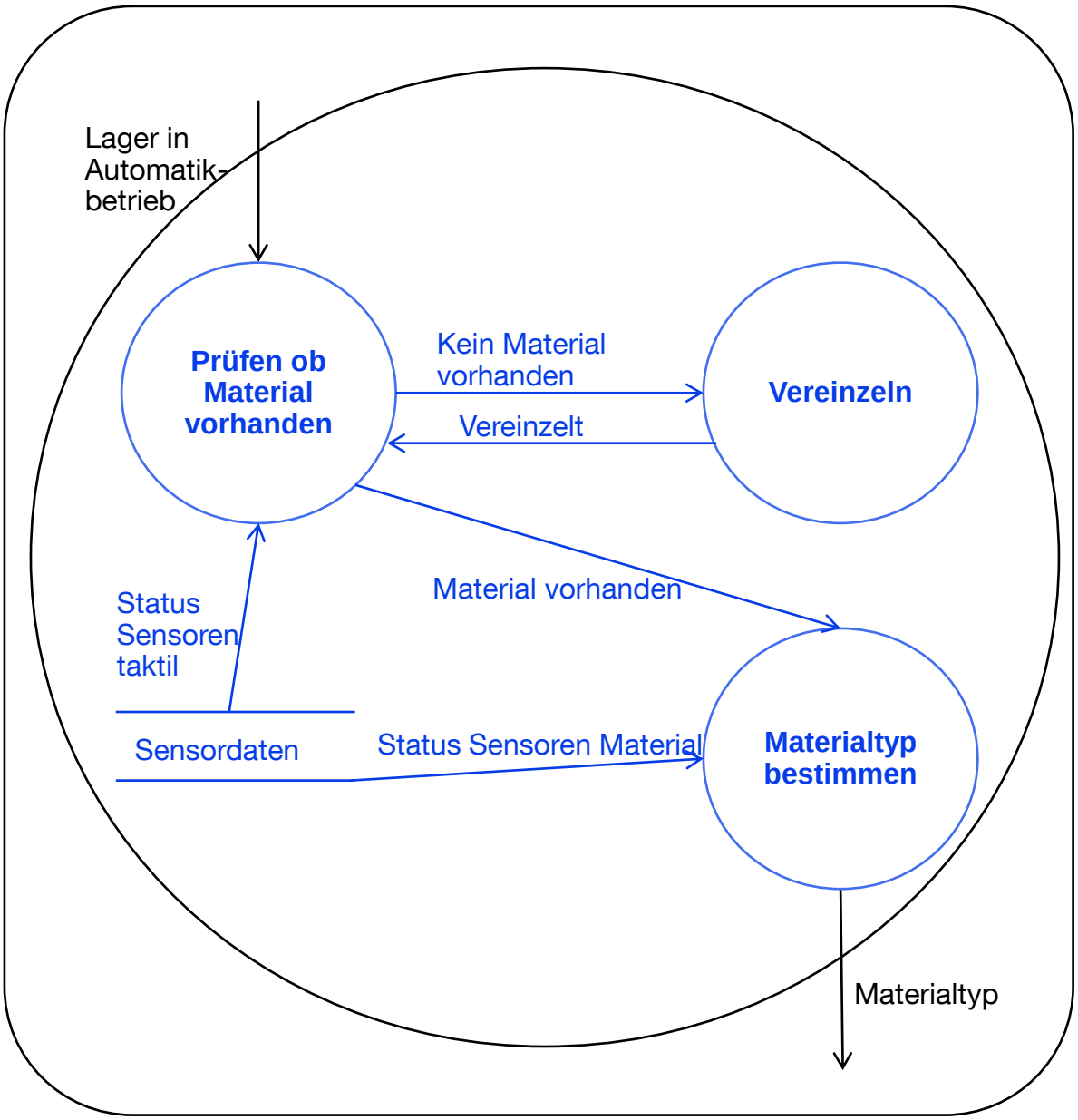
Vorname, Name

Matrikelnummer

Punkte

2. SA/RT

a) Erstellen Sie nun ein Datenflussdiagramm für das Lager. Berücksichtigen Sie dabei die folgenden Prozesse: *Prüfen ob Material vorhanden, Vereinzeln und Material bestimmen*. Der Prozess Prüfen ob Material vorhanden nimmt von Außen die Eingabe *Lager in Automatikbetrieb* entgegen. Er greift außerdem auf den *Datenspeicher Sensordaten* zurück. Der Prozess Vereinzeln nimmt die Eingabe *kein Material vorhanden* entgegen und liefert die Ausgabe *vereinzelt* zurück. Der Prozess Materialtyp bestimmen nimmt die Eingabe *Material vorhanden* entgegen und greift auf die Sensordaten zurück. Er hat außerdem die Ausgabe *Materialtyp*.

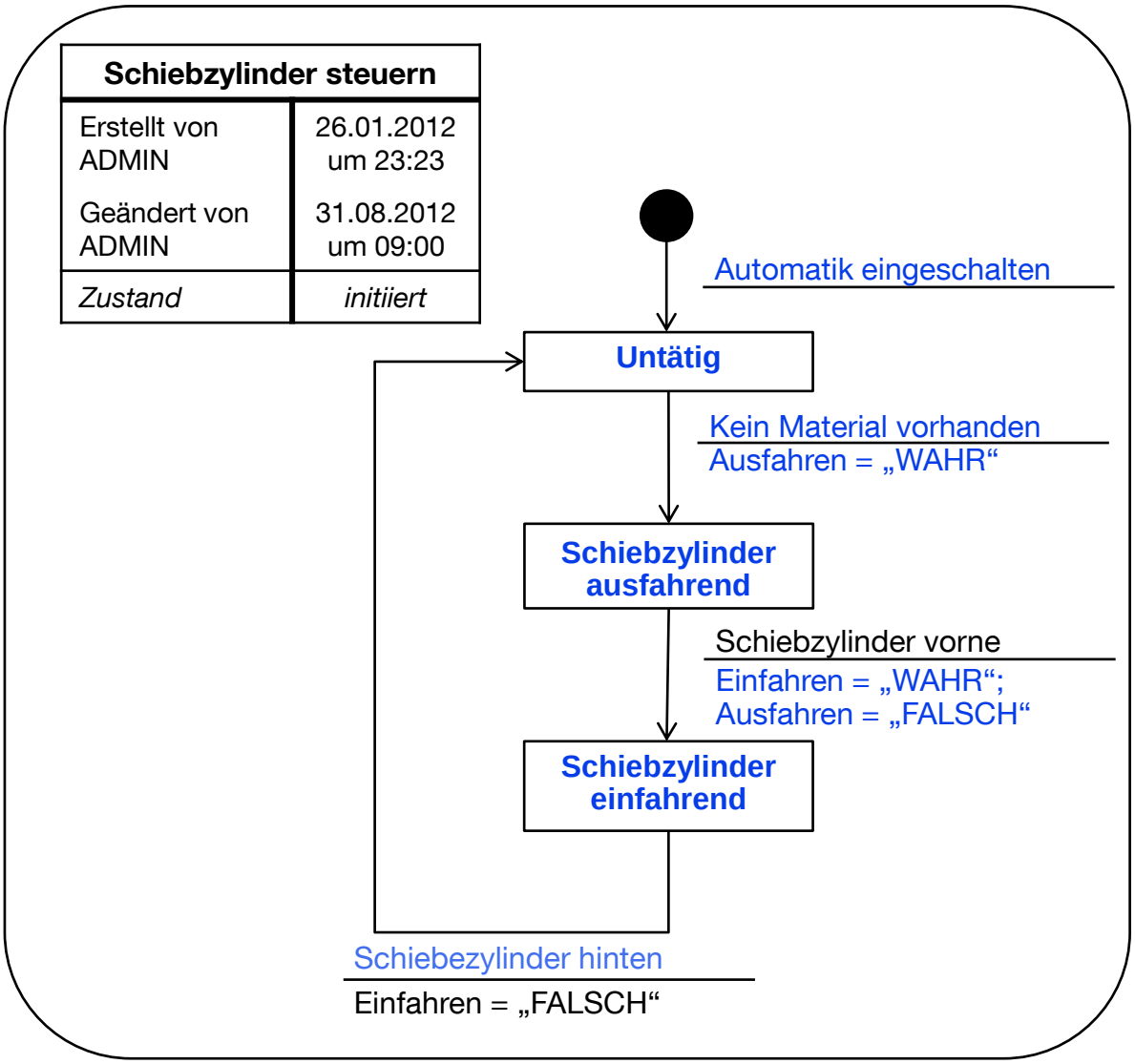


Vorname, Name

Matrikelnummer

Punkte

b) Erstellen Sie nun eine Steuerspezifikation (CSPEC) für den Schiebzylinder. Berücksichtigen Sie dabei die folgenden Zustände: *Untätig, Schiebzylinder ausfahrend* und *Schiebzylinder einfahrend*. Zu Beginn wird bei *einschalten* der *Automatik* der Zustand *Untätig* aktiviert. Unter der Bedingung, dass *kein Material vorhanden* ist, wird die Steuervariable *Ausfahren* auf "wahr" gesetzt und der Zustand *Schiebzylinder ausfahrend* aktiviert. Ist der *Schiebzylinder vorne* wird die Steuervariable *Einfahren* gesetzt, die Steuervariable *Ausfahren* zurückgesetzt und der Zustand *Schiebzylinder einfahrend* aktiviert. Sobald der *Schiebzylinder hinten* ist, wird der Zustand *Untätig* aktiviert. Ergänzen Sie die Zustände und Zustandsübergänge inklusive aller Ereignisse und Bedingungen im Diagramm. Zeichnen Sie keine zusätzliche Zustände ein!



Vorname, Name

Matrikelnummer

Punkte

Aufgabe C: Programmieren in C

Aufgabe C: 96 Punkte

Teil 1: Datentypen, Ein-/Ausgabe und Bool‘sche Algebra

Sie bekommen von einem Autohändler den Auftrag ein Programm zu schreiben, mit dem der Autohändler seine Autos organisieren kann.

a) Legen Sie zunächst je eine Variable an für

- die Leistung (ganzzahlig in PS)
- die Länge der Autos (in Meter, auf zwei Nachkommastellen genau)
- den Motortyp, (entweder „O“ für Ottomotor, oder „D“ für Dieselmotor)

und **initialisieren** Sie die Variablen mit sinnvollen Werten.

```
int iLeistung    =  150;
float fLaenge    =  3.40;
char cMotortyp =  'O';
```

b) Der Händler soll die Möglichkeit haben, über eine Abfrage den Preis des Autos zu bestimmen. Dabei wird der neu eingeführten Variable ***fPreis*** ein Eurowert als Gleitkommazahl zugewiesen. Danach soll der Preis zur Überprüfung auf **zwei Nachkommastellen** genau ausgegeben werden. Füllen Sie die Lücken im Quellcode.

```
float     fPreis = 0;            //Anlegen der Variable
                                 //für den Preis
//Eingabe
printf ("Bitte geben Sie den Preis in Euro ein: ");

if ( __scanf__ ("%f", &fPreis)==0)
{
    printf ("Unzulaessige Eingabe!");
    return 1;
}
//Ausgabe
printf ("\n\nIhr Preis lautet: %.2f    \n", fPreis      );
```

c) Werten Sie folgende Ausdrücke aus, und schreiben Sie das Ergebnis in die rechte Spalte. Berücksichtigen Sie die Veränderung der Variablen in den aufeinanderfolgenden Zeilen. Int-Variablen: A = 2; D = 0; Float-Variablen: B = 0.25; C = π;

(++D || (int)B) == --A	1
!((A&&B) || C)	0
(!A | D)	1

Vorname, Name

Matrikelnummer

Punkte

c) Das unten aufgeführte Programm berechnet den Preis des Autos in Abhängigkeit des Grundpreises, der Anzahl der Extras sowie des Kundenrabattes. Finden Sie und markieren Sie alle 6 Syntaxfehler und markieren Sie zusätzlich die entsprechende Zeile.

Beispiel: `int iHilf = 0` ✗

```
#include <stdio>                                          ✗
int main(){
    float fPreis = 2499.00;      //Grundpreis des Autos
    float fExtra = 125.25;       //Preis pro Extra
    iExtras = 0;                 //Anzahl der Extras       ✗
    int iKundenrabatt = 10;      //Kundenrabatt in Prozent
    int iHilf = 0;

    printf("Anzahl der Extras: ");
    if(scanf("%i",iExtras)==0)                            ✗
    {
        printf("Unzulaessige Eingabe!");
        return 1;
    }
    fPreis += iExtras*fExtra                              ✗
    printf("\nDer 1. Preis betraegt %.2f Euro",fPreis);

    iHilf = iKundenrabatt && iExtras;
    fPreis -= iHilf*(fPreis*(iKundenrabatt/100.00));
    printf(\nDer 2. Preis betraegt %.2f Euro",fPreis);    ✗

    iHilf = (fPreis >= 2500);
    fPreis /= ++iHilf;
    printf("\nDer 3. Preis betraegt i Euro",(int)fPreis);  ✗

return 0;
}
```

d) Geben Sie die Ausgabe des Programms an, wenn bei der Eingabe der Wert „4“ eingegeben wird. Gehen Sie jetzt davon aus, dass der Code von Fehlern bereinigt wurde.

Der 1. Preis betraegt **3000.00** Euro

Der 2. Preis betraegt **2700.00** Euro

Der 3. Preis betraegt **1350** Euro

Vorname, Name

Matrikelnummer

Punkte

Teil 2: Kontrollstrukturen

a) Das Programm soll so erweitert werden, dass der Händler die Farbe eingeben kann, in der das Auto bestellt werden soll. Neben der Standardfarbe Schwarz (S) gibt es gegen einen Aufpreis noch Rot (R). Darüber hinaus kann er entscheiden, ob das Auto als 3-Türer (iBauart=0) oder als 5-Türer (iBauart=1) bestellt werden soll.
Der Preis setzt sich dabei folgendermaßen zusammen:
- Farbe Schwarz und 3 Türen → Grundpreis
- Farbe Schwarz und 5 Türen → Grundpreis + 10%
- Farbe Rot und 3 Türen → Grundpreis + 100€
- Farbe Rot und 5 Türen → (Grundpreis + 10%) + 100€

Gehen Sie davon aus, dass der Code zum Einlesen von Farbe und Anzahl der Türen bereits realisiert wurde. Programmieren Sie nur die **Preisberechnung** für die Variable fPreis unter den genannten Voraussetzungen. Nutzen Sie hierfür **IF-ELSE**.

```
#include <stdio.h>
int main(){
    float fPreis = 2500.00;   //Grundpreis des Autos
    char cFarbe = 'S';        //Farbe des Autos
    int iBauart = 0;      //3-Tuerer (0) oder 5-Tuerer (1)

    //Eingabe der Farbe mit externer Funktion
    cFarbe=cEinlesen();
    //Entscheidung 3- oder 5-Tuerer mit externer Funktion
    iBauart=iEinlesen();

    //Preisberechnung
    if (iBauart && cFarbe == 'S')
        {
            fPreis += 0.10 * fPreis;
        }
        else if (!iBauart && cFarbe == 'R')
        {
            fPreis += 100.00;
         }
         else if (iBauart && cFarbe == 'R')
        {
            fPreis += 100.00 + 0.10 * fPreis;
        }

printf("Preis: %.2f € für Farbe %c und %i Türen\n",\
                                fPreis, cFarbe, 3+iBauart*2 );
return 0;
}
```

Vorname, Name

Matrikelnummer

Punkte

b) Der Händler möchte, dass das Programm eine kleine Preisliste beinhaltet, in der abhängig von der Eingabe des Autohersteller der Preis auf dem Bildschirm ausgegeben wird. Das Einlesen der Benutzereingabe ist bereits von einer externen Funktion realisiert. Wird ein nicht verfügbares Fahrzeug eingegeben, soll das Programm schlicht „Fehler!“ ausgeben. Implementieren Sie diese Funktionsweise mit Hilfe **von SWITCH-CASE**. Legen Sie dafür zunächst ein **Enum** mit den Herstellernamen an. Nutzen Sie dafür folgende Tabelle:

FIAT	10.000€
VW	20.000€
BMW	80.000€

```
#include <stdio.h>

//Anlegen des Variablentyp hersteller als enum

  typedef enum {FIAT, VW, BMW, MERCEDES}HERSTELLER;

int main(){
    HERSTELLER hersteller;   //Variable vom Typ HERSTELLER
                             //anlegen
    //Eingabe des Herstellers mit externer Funktion
    hersteller=hEinlesen();

//Ausgabe des Preises abhaenging vom Hersteller
//z.B. 10.000€ für Fiat

  switch( hersteller)
      {
            case FIAT :     printf( „10.000€" );
                            break;
            case VW:        printf( „20.000€" );
                            break;
            case BMW:       printf( „80.000€" );
                            break;
            default :       printf( „Fehler!");
      }

return 0;
}
```

Vorname, Name

Matrikelnummer

Punkte

Teil 3: Erweiterte Datentypen

a) Der Händler möchte alle Mercedes-Benz, die er bestellen soll, in einer kompakten Form zusammenfassen. Es soll also in einem Array eine Liste von 5 Mercedes-Benz festgehalten werden. Dabei wird nur der Fahrzeugklassen-Buchstabe (z.B. ‚C‘ für Mercedes C-Klasse) repräsentativ für das Auto in das Array geschrieben.
Deklarieren Sie ein Array von geeignetem Typ und geeigneter Länge und benennen Sie es sinnvoll. **Initialisieren** Sie dann den ersten Eintrag mit ‚S‘.

```
char cMercedesListe[5];

cMercedesListe[0] = 'S';
```

b) Das Array soll jetzt um eine Dimension erweitert werden um neben der Mercedes-Benz „Klasse“ auch noch den Motortyp festzulegen. Für den Motortyp kann dabei ‚D‘ als Dieselmotor oder ‚O‘ als Ottomotor eingesetzt werden.
Deklarieren Sie ein neues Array von geeignetem Typ und geeigneter Länge und benennen Sie es sinnvoll. **Initialisieren** Sie dann das erste Element des ersten Eintrags mit ‚S‘ und das zweite Element des ersten Eintrags mit ‚D‘.

```
char cMercedesListe[5][2];

cMercedesListe[0][0] = 'S';

cMercedesListe[0][1] = 'D';
```

Vorname, Name _____ Matrikelnummer _____

Punkte

Teil 4: Schleifen

a) Zur Vervollständigung vorgegeben ist ein Programm zum Einlesen von zwei geometrischen Vektoren in die Variablen iAVektor1 und iAVektor2, welche jeweils die drei Raumrichtungen x,y,z enthalten sollen. Vervollständigen Sie nun den Rest des Codes so, dass die drei Elemente des Vektor iAVektor1 vom Benutzer eingelesen werden können. Verwenden Sie hierzu eine **FOR-Schleife**.

Hinweis : Um nicht für jede Raumrichtung einen eigenen printf-Befehl verwenden zu müssen, wird ein Ascii-Zeichen übergeben, welches mit der Schleife hochzählt. Das Ascii-Zeichen ‚x' hat in der ASCII-Standardtabelle den Wert 120.

```
#include <stdio.h>

int main()
{
int i=0;
int iAVektor1[3]={0,0,0};
int iAVektor2[3]={0,0,0};
int iAErgebnis[3];

for(i=0;i<=2;i++)
oder: for(i=0;i<3;i++)
_____________________
    {
        printf("\nBitte geben Sie die %c-Richtung des\
               1.Vektor ein:", 120+i                    );

                        &iAVektor1[i]
        if(scanf("%i", oder: iAVektor1+i              )==0)
            {printf("\nFehler Eingabe!");return 1;}
    }
    printf("\n");
return 0;
}
```

Vorname, Name

Matrikelnummer

Punkte

b) Weiterhin soll noch die Additionsberechnung durchgeführt werden. Dazu werden die Elemente der jeweiligen Raumrichtung von Vektor „iAVektor1“ und „iAVektor2“ in „iAErgebnis“ (bereits initialisiert) addiert. Gehen Sie davon aus, dass nun beide Vektoren eingelesen worden sind. Verwenden Sie nun hierbei eine **DO-WHILE-Schleife** oder eine **WHILE-Schleife**.

```
i=0;
  "Do" oder "while(i<3)"

    {
     iAErgebnis[i]=iAVektor1[i]+iAVektor2[i];

     i++;  oder i=i+1; oder i+=1;

    }while(i<3); oder entfällt falls while oben
```

Teil 5: Zeiger

Gegeben sei ein kurzes C-Programm, welches ein Array mit vier Elementen und die Zeigervariable „pizeiger“ initialisiert. Die darauf folgenden Befehle führen jeweils zur einer Manipulation des Zeigers, sowie der Werte im Array. Geben Sie die veränderten Werte des Arrays nach dem Ausführen des Codes an. Kreuzen Sie in der mittleren Spalte an wo sich der Zeiger am Ende des Codes befindet.

```
#include <stdio.h>
int main ()
{
   int *pizeiger=NULL;
   int Ai[4]={3,9,4,10};
   pizeiger=Ai;
   *pizeiger=*pizeiger+2*2;
   pizeiger=&Ai[1]+1;
   *(++pizeiger)+=*(pizeiger-1);
   *(pizeiger-1)=Ai[1]%2;
   return 0;
}
```

Array-Element	Zeiger	Wert des Element
Ai[0]		7
Ai[1]		9
Ai[2]		1
Ai[3]	X	14

Vorname, Name

Matrikelnummer

Punkte

Teil 6: Funktionen

Gegeben sei ein kurzes C-Programm, welches zwei Integer-Variablen **unterschiedlich** miteinander multiplizieren soll. Der Einfachheit halber wurde auf das Einlesen verzichtet. Beachten Sie daher die **Initialisierung der Variablen** in der Main-Funktion. Die Funktionalität des Mulitiplizierens wurde einmal mit **Call-by-reference** und einmal mit **Call-by-value** umgesetzt. Vervollständigen Sie die zugehörigen Funktionsdeklarationen. Geben Sie die Ausgabe des Programms im unteren Lösungsfeld an.

```
#include <stdio.h>

//Call-by-value

___int_____ Funktion1(_int iVar1, int iVar2_____)
{
  return iVar1*iVar2;
}

//Call-by-reference

___void_____ Funktion2(_int* iZahl1, int* iZahl2, int* i Erg_____)
{
   *iErg=2*(*iZahl1)*(*iZahl2);   Auch: int *iZahl1,…
}
int main()
{
    int iA=3,iB=2, iRes=0;

    printf("\n1. Ergebnis: %i",Funktion1(iA,iB));

    Funktion2(&iA,&iB,&iRes);
    printf("\n\n2. Ergebnis: %i",!iRes);
    return 0;
}
```

```
1. Ergebnis: _6____

2. Ergebnis: _0____
```

Vorname, Name

Matrikelnummer

Punkte

Teil 7: Bubble-Sort-Algorithmus

Gegeben sei ein kurzes C-Programm, welches ein Integer-Array der Größe seiner Zahlen nach sortieren soll. Vervollständigen Sie die **Bedingung der WHILE-Schleife**, welche abbricht wenn alle nötigen Vertauschvorgänge abgeschlossen sind. Vervollständigen Sie weiterhin **die IF-Bedingung**, welche zur Sortierreihenfolge „**vom ersten Element absteigend**“ führt. Abschließend schreiben Sie den Code, welcher die Vertauschung der beiden geprüften Elemente durchführt. Benutzen Sie hierfür die **Variable iTemp**.

```
#include <stdio.h>
void Sortieren(int* iArray,int iGroesse)
{
    int i=0,k=1,iTemp=0;

  //Von vorne Starten bis keine Tauschaktionen
  //mehr durchgeführt worden sind.

  while( k!=0 oder k>0 )
  {
    k=0;
    //Absteigend Sortieren, d.h. höchste Zahl zuerst.
    while(i<iGroesse-1)
          {
            if(iArray[i]  <  iArray[i+1])

            //i. und (i+1). Zahl in iArray tauschen
            //falls IF-Bedingung erfüllt ist.
                    {  iTemp=iArray[i+1];
                       iArray[i+1]=iArray[i];
                       iArray[i]=iTemp;

                       //Tauschaktion mitzählen
                       k++;
                    }
          i++;
        }
    i=0;
  }
}
int main()
{
    int iArray[]={1,2,3};
    int i=0;
    //Funktionsaufruf um iArray zu Sortieren
    Sortieren(iArray,3);
    return 0;
}
```

oder:

iTemp=iArray[i];

iArray[i]=iArray[i+1];

iArray[i+1]=iTemp;

Vorname, Name

Matrikelnummer

Punkte

Teil 8: Verkettete Liste

Gegeben ist der Ausschnitt eines C-Programm mit einer verketteten Liste. Der Code zum einfügen neuer Elemente ist nicht dargestellt. Vervollständigen Sie die Funktion „Ausgabe“, welche es ermöglicht gezielt einzelne Elemente der verketteten Listen aufzurufen und anzuzeigen. Der **zugehörige Aufruf** zum Anzeigen des 5. Element wurde beispielhaft in der „main“-Funktion implemenentiert.

```
(...)
typedef struct LISTE_s{
  char sName[60];
  struct LISTE_s *pNext;
}FAHRZEUG;

typedef struct{
  int iAnzahl;
  FAHRZEUG *pFirst;
}KOPF;

__FAHRZEUG*__ Ausgabe(__KOPF*__ Liste,__int__ iStelle)
{
  int ia=0;
  FAHRZEUG* pTemp=NULL;
// Zeiger umbiegen bis die gesuchte Stelle erreicht ist.

          __pTemp=Liste->pFirst;__

          __for(ia=1;ia<iStelle;ia++)__

          {

          __pTemp=pTemp->pNext;__

          }
return pTemp;
}
(...)

int main()
{
 //Variablendeklaration
  int i=0; KOPF Liste={0,NULL}; FAHRZEUG *pAusgabe=NULL;

//Ausgabe des 5. Element
    pAusgabe=Ausgabe(&Liste, 5);
    printf("%5.Fahrzeug: %s\n",pAusgabe->sName);
(...)
}
```