

Abiturprüfung 2011

Hinweise zur Korrektur und Bewertung
der Abiturprüfungsarbeiten in

Informatik

Nicht für den Prüfling bestimmt

Da es sich bei Informatik um ein neues Abiturprüfungsfach handelt, sind die Lösungs- und Korrekturhinweise verhältnismäßig ausführlich gehalten. Dennoch können nicht alle Lösungsalternativen dargestellt werden. Nicht genannte, aber `g l e i c h w e r t i g e` Lösungswege und Begründungsansätze sind `g l e i c h b e r e c h t i g t`.

Die in den Lösungshinweisen gewählte objektorientierte Programmiersprache ist Java.

Die Bewertung der erbrachten Prüfungsleistungen hat sich an der bei jeder Teilaufgabe am linken Rand des Angabenblattes vermerkten, maximal erreichbaren Zahl von Bewertungseinheiten (BE) zu orientieren.

Umrechnung der erreichten Bewertungseinheiten in Notenpunkte

Die insgesamt erreichten Bewertungseinheiten werden nach der folgenden Tabelle in Notenpunkte umgesetzt:

Notenpunkte	Notenstufen	Bewertungseinheiten	Intervalle in %
15	+1	120 ... 115	15
14	1	114 ... 109	
13	1–	108 ... 103	
12	+2	102 ... 97	15
11	2	96 ... 91	
10	2–	90 ... 85	
9	+3	84 ... 79	15
8	3	78 ... 73	
7	3–	72 ... 67	
6	+4	66 ... 61	15
5	4	60 ... 55	
4	4–	54 ... 49	
3	+5	48 ... 41	20
2	5	40 ... 33	
1	5–	32 ... 25	
0	6	24 ... 0	20

Informatik

Aufgabe Inf1.I

Aufgabe	BE	Hinweise
1. a)	10	<pre> classDiagram class TISCH { tischnr anzahlGaeste anlass bestellungAnfuegen(gerichtID) offeneRechnungZeigen() zahlen() } class POSTEN { status ID } class GERICHT { ID bezeichnung preis anzahlVorhanden } TISCH "1" -- "n" POSTEN : bestellt > POSTEN "n" -- "1" GERICHT : enthält > </pre>
b)	5	<pre> classDiagram class LEISTUNG { <<abstract>> ID bezeichnung preis anzahlVorhanden drucken() {abstract} istVorhanden() } class GERICHT { drucken() } class GETRAENK { istAlkoholisch drucken() } LEISTUNG < -- GERICHT LEISTUNG < -- GETRAENK </pre> Hinweis: Möglich wäre auch, <i>drucken()</i> in der Klasse LEISTUNG nicht abstrakt zu definieren und in der Klasse GETRAENK bzw. GERICHT geeignet zu überschreiben.
c)	13	In der abstrakten Klasse LEISTUNG: <pre> boolean istVorhanden() { return (anzahlVorhanden > 0); } abstract void drucken(); </pre>

In der Klasse GETRAENK:

```
void drucken() {
    // istAlkoholisch ist vom Datentyp boolean
    if (istAlkoholisch) {
        System.out.println(bezeichnungGeben() + " "
            + preisGeben() + " EUR A");
    }
    else {
        System.out.println(bezeichnungGeben() + " "
            + preisGeben() + " EUR");
    }
}
```

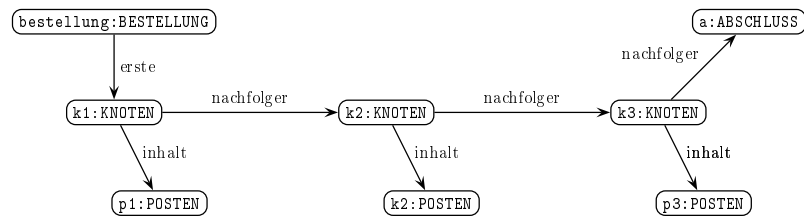
In der Klasse GERICHT:

```
void drucken() {
    System.out.println(bezeichnungGeben()+ " "
        + preisGeben() + " EUR");
}
```

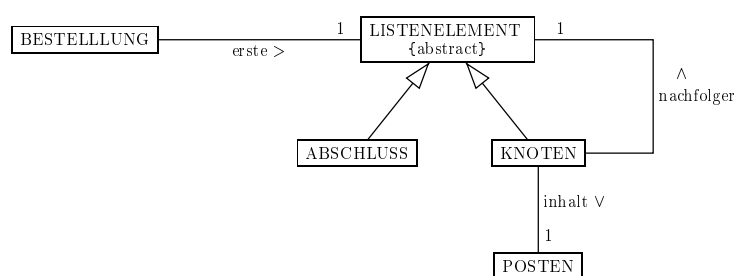
Hinweis: Bei der Bewertung kann auch der direkte Zugriff auf die Attribute *bezeichnung* und *preis* anstelle der zugehörigen Geben-Methoden akzeptiert werden.

d)

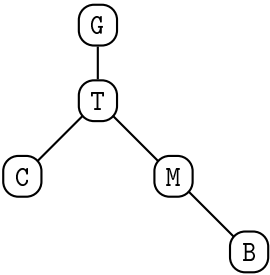
11



Hinweis: Möglich ist auch eine Nennung der Referenzattribute innerhalb der Objekte.



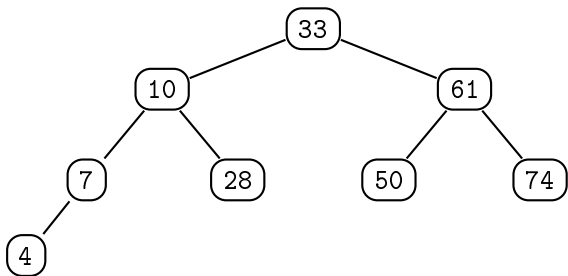
e)	13	In Klasse BESTELLUNG: <pre>double rechnungsbetragGeben() { return erste.rechnungsbetragGeben(); }</pre> In Klasse LISTENELEMENT: <pre>abstract double rechnungsbetragGeben();</pre> In Klasse KNOTEN: <pre>double rechnungsbetragGeben() { if (inhalt.istAusgeliefert()) return inhalt.preisGeben() + nachfolger.rechnungsbetragGeben(); else return nachfolger.rechnungsbetragGeben(); }</pre> In Klasse ABSCHLUSS: <pre>double rechnungsbetragGeben() { return 0; }</pre>																																																	
f)	3	<pre>SELECT Bezeichnung, Preis FROM speisen WHERE AnzahlVorhanden > 10</pre>																																																	
g)	4	(1) Verbindung zur Datenbank herstellen (2) Abfrage an die Datenbank schicken (3) Abfrageergebnis zeilenweise auslesen und z. B. auf dem Bildschirm in einer Tabelle ausgeben (4) Verbindung zur Datenbank beenden																																																	
2. a)	6	Es handelt sich um einen ungerichteten, bewerteten Graph. (In der Adjazenzmatrix sind die Städte mit ihren Anfangsbuchstaben bezeichnet.) <table border="1"><tr><td></td><td>M</td><td>T</td><td>C</td><td>S</td><td>G</td><td>B</td></tr><tr><td>M</td><td></td><td>2</td><td></td><td></td><td></td><td>9</td></tr><tr><td>T</td><td>2</td><td></td><td>6</td><td>8</td><td>8</td><td></td></tr><tr><td>C</td><td></td><td>6</td><td></td><td>3</td><td>3</td><td></td></tr><tr><td>S</td><td></td><td>8</td><td>3</td><td></td><td></td><td></td></tr><tr><td>G</td><td></td><td>8</td><td>3</td><td></td><td></td><td>12</td></tr><tr><td>B</td><td>9</td><td></td><td></td><td></td><td>12</td><td></td></tr></table> (Die leeren Felder könnten z. B. auch mit -1, die Felder der Hauptdiagonalen auch mit 0 belegt werden.)		M	T	C	S	G	B	M		2				9	T	2		6	8	8		C		6		3	3		S		8	3				G		8	3			12	B	9				12	
	M	T	C	S	G	B																																													
M		2				9																																													
T	2		6	8	8																																														
C		6		3	3																																														
S		8	3																																																
G		8	3			12																																													
B	9				12																																														

b)	11	Die Klasse GRAPH benötigt zusätzlich das Attribut <i>besucht</i> vom Typ <code>boolean[]</code>: Ein Feldelement hat den Wert <i>true</i>, wenn der entsprechende Knoten schon besucht/bearbeitet wurde, sonst hat es den Wert <i>false</i>. <pre> Methode tiefensuche (int startKnotenNr) zähle i von 0 bis anzahlKnoten-1 besucht[i] = false //Zuweisung endeZähle besuchen(startKnotenNr) endeMethode Methode besuchen (int knotenNr) //Nur etwas tun, wenn der Knoten noch nicht besucht worden ist wenn (nicht besucht[knotenNr]) dann //aktiven Knoten auf besucht setzen und Stadt auf //der Konsole ausgeben besucht[knotenNr] = true drucken(knotenNr) //Nachbar- bzw. Nachfolgeknoten besuchen: zähle i von 0 bis anzahlKnoten-1 wenn (matrix[knotenNr][i] > 0) dann besuchen(i) endeWenn endeZähle endeWenn endeMethode </pre>
c)	4	Es existiert immer noch ein Zyklus. Entferne z. B. die Kante C–G.  <pre> graph TD G((G)) --- T((T)) T --- C((C)) T --- M((M)) M --- B((B)) </pre>
	80	

Informatik

Aufgabe Inf1.II

Aufgabe	BE	Hinweise																																				
1. a)	11	<table><tr><th>KUNDE</th><td></td><th>BUCHUNG</th><td></td><th>FLUG</th></tr><tr><td>kundennummer</td><td rowspan="3">macht > 1-----n</td><td>buchungsnr</td><td rowspan="3">enthält > n-----1</td><td>flugnr</td></tr><tr><td>name</td><td>datum</td><td>datum</td></tr><tr><td>e-mail-adresse</td><td>anzahlPlaetze</td><td>startzeit</td></tr><tr><td>buchungHinzufuegen(buchung)</td><td></td><td>kategorie</td><td></td><td>flughafen</td></tr><tr><td>buchungStornieren(buchung)</td><td></td><td>gesamtpreis</td><td></td><td>zielort</td></tr><tr><td>alleBuchungenZeigen()</td><td></td><td>preisAktualisieren()</td><td></td><td>airline</td></tr><tr><td></td><td></td><td></td><td></td><td>datenZeigen()</td></tr></table> Hinweis: Die angeführten Methoden sind als Vorschlag zu sehen. Nicht alle müssen in einer korrekten Lösung in dieser Form vorkommen.	KUNDE		BUCHUNG		FLUG	kundennummer	macht > 1-----n	buchungsnr	enthält > n-----1	flugnr	name	datum	datum	e-mail-adresse	anzahlPlaetze	startzeit	buchungHinzufuegen(buchung)		kategorie		flughafen	buchungStornieren(buchung)		gesamtpreis		zielort	alleBuchungenZeigen()		preisAktualisieren()		airline					datenZeigen()
KUNDE		BUCHUNG		FLUG																																		
kundennummer	macht > 1-----n	buchungsnr	enthält > n-----1	flugnr																																		
name		datum		datum																																		
e-mail-adresse		anzahlPlaetze		startzeit																																		
buchungHinzufuegen(buchung)		kategorie		flughafen																																		
buchungStornieren(buchung)		gesamtpreis		zielort																																		
alleBuchungenZeigen()		preisAktualisieren()		airline																																		
				datenZeigen()																																		
b)	14	<pre>public class Kunde { //Attribute private int kundennummer; private String name; private String email; private Buchung[] buchungen; private int anzahlBu; //Konstruktor public Kunde(int kn, String n, String em){ kundennummer = kn; name = n; email = em; buchungen = new Buchung[20]; anzahlBu = 0; } //Methoden public void buchungHinzufuegen(Buchung neueBu){ if(anzahlBu < 20){ buchungen[anzahlBu] = neueBu; anzahlBu = anzahlBu + 1; } else { for(int i = 0; i < anzahlBu - 1; i++){ //oder i < 19 buchungen[i] = buchungen[i+1]; } buchungen[19] = neueBu; } } }</pre>																																				

c)	2	Ein Feld hat beispielsweise folgende Nachteile: • Die Länge des Feldes ist von vornherein begrenzt. • Für nicht benutzte Feldelemente wird unnötig Speicherplatz reserviert. • Der Aufwand beim Entfernen eines Elementes ist relativ hoch, weil beispielsweise alle nachfolgenden Feldelemente um eine Position nachgerückt werden müssen. • Der Aufwand beim Sortieren (z. B. nach Flugdatum etc.) ist möglicherweise recht hoch.														
d)	8	Mögliche Lösung:  Man müsste maximal vier Knoten (inkl. Wurzel) besuchen. Der Postorder-Durchlauf lautet in diesem Fall: 4 – 7 – 28 – 10 – 50 – 74 – 61 – 33.														
e)	8	<table><tr><th>Anzahl der Ebenen</th><th>maximale Knotenzahl</th></tr><tr><td>1</td><td>1</td></tr><tr><td>2</td><td>1 + 2 = 3</td></tr><tr><td>3</td><td>3 + 4 = 7</td></tr><tr><td>4</td><td>7 + 8 = 15</td></tr><tr><td>⋮</td><td></td></tr><tr><td>k Ebenen</td><td>2^k – 1</td></tr></table> $\begin{aligned} 2^k - 1 &\geq 10000000 \\ k \cdot \ln 2 &\geq \ln 10000001 \\ k &\geq 23, 25 \dots \\ \implies k_{min} &= 24 \end{aligned}$	Anzahl der Ebenen	maximale Knotenzahl	1	1	2	1 + 2 = 3	3	3 + 4 = 7	4	7 + 8 = 15	⋮		k Ebenen	2 ^k – 1
Anzahl der Ebenen	maximale Knotenzahl															
1	1															
2	1 + 2 = 3															
3	3 + 4 = 7															
4	7 + 8 = 15															
⋮																
k Ebenen	2 ^k – 1															

		Es müssen maximal 24 Knoten besucht werden. Ansatz: Die maximale Anzahl an Knoten, die ein Binärbaum mit k Ebenen haben kann, muss mindestens so groß sein wie die vorhandene Knotenanzahl. Wären die Kundendaten hingegen in Form einer verketteten Liste gespeichert, müsste man im ungünstigsten Fall alle 10 Millionen Knoten besuchen, um das gewünschte Objekt zu finden. Suchanfragen können daher bei balancierten Binärbäumen wesentlich schneller durchgeführt werden.
2. a)	6	<pre> graph LR w["w: WARTESCHLANGE"] -- erster --> k1["k1: DATENKNOTEN"] k1 -- inhalt --> f1["f1: FLUG"] k1 -- nächster --> k2["k2: DATENKNOTEN"] k2 -- inhalt --> f2["f2: FLUG"] k2 -- nächster --> k3["k3: DATENKNOTEN"] k3 -- inhalt --> f3["f3: FLUG"] k3 -- nächster --> a["a: ABSCHLUSS"] </pre> Hinweis: Möglich ist auch eine Nennung der Referenzattribute innerhalb der Objekte.
b)	14	In der Klasse WARTESCHLANGE: <pre> public int passagierSummeGeben(){ return erster.passgesamtGeben(); } </pre> In der Klasse FLUG: <pre> public int passagierzahlGeben(){ return passagierzahl; } </pre> In der Klasse LISTENELEMENT: <pre> public abstract int passgesamtGeben(); </pre> In der Klasse DATENKNOTEN: <pre> public int passgesamtGeben(){ return inhalt.passagierzahlGeben() + naechster.passgesamtGeben(); } </pre> In der Klasse ABSCHLUSS: <pre> public int passgesamtGeben(){ return 0; } </pre>

c)	3	Es handelt sich um das Entwurfsmuster „Kompositum“. In der Softwareentwicklung werden Softwaremuster verwendet, um bei ähnlichen Aufgabenstellungen auf bewährte und wiederverwendbare Lösungsansätze zurückgreifen zu können, was wiederum die Wiederverwendung des erstellten Codes erleichtert. So spart man Programmieraufwand und damit Kosten.																																				
3. a)	8	<table border="1"><thead><tr><th></th><th>Mün.</th><th>Frankf.</th><th>Amst.</th><th>Oslo</th><th>Trondh.</th></tr></thead><tbody><tr><td>München</td><td>0</td><td>1,5</td><td>2</td><td>3</td><td></td></tr><tr><td>Frankfurt</td><td>1,5</td><td>0</td><td>1</td><td>2</td><td>3,5</td></tr><tr><td>Amsterdam</td><td>2</td><td>1</td><td>0</td><td></td><td>2,5</td></tr><tr><td>Oslo</td><td>3</td><td>2</td><td></td><td>0</td><td>1</td></tr><tr><td>Trondheim</td><td></td><td>3,5</td><td>2,5</td><td>1</td><td>0</td></tr></tbody></table> Auf die Angabe der 0 in der Diagonale kann auch verzichtet werden. <pre>graph TD; Trondheim --- 2,5 Amsterdam; Trondheim --- 3,5 Frankfurt; Trondheim --- 1 Oslo; Amsterdam --- 1 Frankfurt; Amsterdam --- 2 München; Oslo --- 2 Frankfurt; Oslo --- 3 München; Frankfurt --- 1,5 München;</pre>		Mün.	Frankf.	Amst.	Oslo	Trondh.	München	0	1,5	2	3		Frankfurt	1,5	0	1	2	3,5	Amsterdam	2	1	0		2,5	Oslo	3	2		0	1	Trondheim		3,5	2,5	1	0
	Mün.	Frankf.	Amst.	Oslo	Trondh.																																	
München	0	1,5	2	3																																		
Frankfurt	1,5	0	1	2	3,5																																	
Amsterdam	2	1	0		2,5																																	
Oslo	3	2		0	1																																	
Trondheim		3,5	2,5	1	0																																	
b)	6	Möglich ist z. B. die Tiefensuche: Bei der Tiefensuche werden ausgehend von einem Startknoten s alle von s aus erreichbaren Knoten besucht. Hierbei wird zunächst in die Tiefe gegangen, d. h. es wird ein Nachbarknoten (unter Berücksichtigung der Speicherreihenfolge) eines bereits erreichten Knotens weiter verfolgt, bevor die anderen Nachbarn dieses Knotens besucht werden. Die Orte werden dann in dieser Reihenfolge besucht: Amsterdam – München – Frankfurt – Oslo – Trondheim																																				
	80																																					

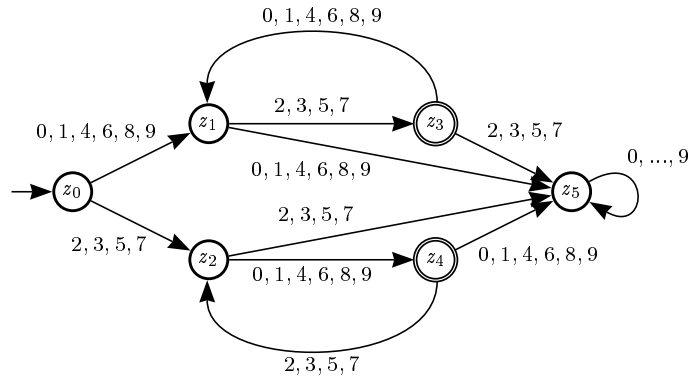
Informatik

Aufgabe Inf2.III

Aufgabe	BE	Hinweise																														
1. a)	3	<table><tr><th>Befehl</th><th>A</th><th>BZ</th><th>101</th><th>102</th></tr><tr><td>(Anfangszustand)</td><td>0</td><td>1</td><td>5</td><td>7</td></tr><tr><td>LOAD 102</td><td>7</td><td>2</td><td>5</td><td>7</td></tr><tr><td>SUB 101</td><td>2</td><td>3</td><td>5</td><td>7</td></tr><tr><td>JGE 6</td><td>2</td><td>6</td><td>5</td><td>7</td></tr><tr><td>END</td><td>2</td><td>7</td><td>5</td><td>7</td></tr></table>	Befehl	A	BZ	101	102	(Anfangszustand)	0	1	5	7	LOAD 102	7	2	5	7	SUB 101	2	3	5	7	JGE 6	2	6	5	7	END	2	7	5	7
Befehl	A	BZ	101	102																												
(Anfangszustand)	0	1	5	7																												
LOAD 102	7	2	5	7																												
SUB 101	2	3	5	7																												
JGE 6	2	6	5	7																												
END	2	7	5	7																												
b)	3	i) Zu Beginn: a = 15; b = 5; am Ende: q = 3; r = 0; ii) Zu Beginn: a = 13; b = 5; am Ende: q = 2; r = 3; Berechnet wird der ganzzahlige Quotient q und der Teilungsrest r bei der Division von a durch b																														
c)	7	1: LOAD 101 2: SUB 102 -- a - b 3: JLT 9 -- falls a < b 4: STORE 101 -- a = a - b 5: DLOAD 1 6: ADD 103 7: STORE 103 -- q = q + 1 8: JUMP 1 9: LOAD 101 10: STORE 104 -- r = a 11: END																														
2.	8	Zeit = Std ':' Min ':' Sek ; Std = 2erZiffer 10erZiffer '2' 4erZiffer ; Min = 6erZiffer 10erZiffer ; Sek = Min [',' 10erZiffer [10erZiffer]] ; (* auch x,00 möglich *) 2erZiffer = '0' '1' ; 4erZiffer = 2erZiffer '2' '3' ; 6erZiffer = 4erZiffer '4' '5' ; 10erZiffer = 6erZiffer '6' '7' '8' '9'; Bemerkung: Als vollständige Lösung könnte z.B. auch noch gelten: Zeit = Std ':' Min ':' Sek ; Std = '0' '0' '0' '1' .. '2' '3' ; Min = '0' '0' '0' '1' .. '5' '9' ; Sek = Min [',' Ziffer [Ziffer]] ; Ziffer = '0' '1' .. '9' ;																														

3. a)

6



Hinweis: Der „Fehlerzustand“ z_5 ist für eine korrekte Lösung nicht unbedingt erforderlich.

b)

3

Beispiel 1: 4567

Zustand	Zeichen	neuer Zustand
z_0	4	z_1
z_1	5	z_3
z_3	6	z_1
z_1	7	z_3

z_3 ist ein Endzustand, also wird die Ziffernfolge akzeptiert.

Beispiel 2: 123456

Zustand	Zeichen	neuer Zustand
z_0	1	z_1
z_1	2	z_3
z_3	3	z_5
z_5	4	z_5
z_5	5	z_5
z_5	6	z_5

z_5 ist kein Endzustand, also wird die Ziffernfolge nicht akzeptiert.

Falls der Fehlerzustand z_5 nicht modelliert wurde:

Zustand	Zeichen	neuer Zustand
z_0	1	z_1
z_1	2	z_3
z_3	3	nicht def.

Die Zeichenkette ist nicht vollständig abgearbeitet, also wird die Ziffernfolge nicht akzeptiert.

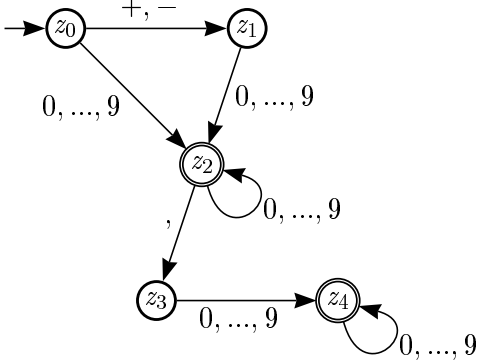
Hinweis: Statt der Darstellung in Tabellenform sind auch Zustandsfolgen möglich.

c)	10	Hinweise: Der Lösungsvorschlag orientiert sich an Java-Syntax. Statt der vorgegebenen Klassen und Methoden dürfen auch äquivalente Klassen und Methoden der Bibliotheken der gewählten Programmiersprache verwendet werden. Laut Aufgabenstellung genügt, es die von zwei Zuständen ausgehenden Übergänge zu implementieren, also z. B.: <pre> private void zustandWechseln(ZEICHEN eingabe) { switch (zustand) { case 0: { switch (eingabe) { case 'N': zustand = 1; break; case 'P': zustand = 2; break; } } break; ... case 3: { switch (eingabe) { case 'N': zustand = 1; break; case 'P': zustand = 5; break; } } break; ... } } public boolean folgePruefen(ZEICHENKETTE code) { for (int i = 0; i < code.laenge(); i++) { zustandWechseln(code.zeichenAnPosition(i)); } return (zustand==3 zustand==4); } </pre> Beispiel für eine vollständige Lösung der Methode <i>zustandWechseln</i>: <pre> private void zustandWechseln(ZEICHEN eingabe) { switch (zustand) { case 0: { switch (eingabe) { case 'N': zustand = 1; break; case 'P': zustand = 2; break; } } break; case 1: { switch (eingabe) { case 'N': zustand = 5; break; case 'P': zustand = 3; break; } } break; } } </pre>
----	----	---

```
case 2: {
    switch (eingabe) {
        case 'N': zustand = 4; break;
        case 'P': zustand = 5; break;
    }
} break;
case 3: {
    switch (eingabe) {
        case 'N': zustand = 1; break;
        case 'P': zustand = 5; break;
    }
} break;
case 4: {
    switch (eingabe) {
        case 'N': zustand = 5; break;
        case 'P': zustand = 2; break;
    }
} break;
}
```

Informatik

Aufgabe Inf2.IV

Aufgabe	BE	Hinweise
1. a)	2	2500, -13 und +3 werden akzeptiert, 0 wird nicht akzeptiert. Bewertungshinweis: Pro Fehler wird ein Punkt abgezogen.
b)	4	<code>zahl = ['+', '-' '0' ... '9'];</code> <code>ziffernicht0 = '1' '2' ... '9';</code> <code>ziffer = '0' ziffernicht0;</code>
c)	5	
2. a)	5	98: 18 99: 0 100: 6
b)	4	<pre> a = 0 while b > 0 a = a + c b = b - 1 </pre> Hinweis: Auch äquivalente Lösungen, z.B. unter Verwendung der Wiederholung mit Anfangsbedingung, sind zulässig.

c)	13	<pre> 1 loadi 1 // Startwert 1 für Potenz 2 store 97 3 load 97 // Zwischenergebnis des Potenzwerts wird nach // Zelle 99 übertragen 4 store 99 5 loadi 0 // von Zelle 5 bis Zelle 13: Multiplikation von // Zellwert 99 und 100 6 store 98 7 load 98 8 add 100 9 store 98 10 load 99 11 dec 12 store 99 13 jg 7 14 load 98 // Produkt wird als neues Zwischenergebnis // nach 97 übertragen 15 store 97 16 load 96 // Zähler (Exponent) wird um 1 verringert 17 dec 18 store 96 19 jg 3 // Rücksprung (Wiederholung), falls Zähler // noch nicht 0 20 end </pre> Speicherzellenbelegung: Zu Beginn muss a in Speicherzelle 100, n in Speicherzelle 96 stehen; am Ende steht der Potenzwert in Speicherzelle 97. Hinweise: Lösungsidee: Zur Berechnung der Potenz wird das in Zelle 97 stehende Zwischenergebnis n mal mit dem in Speicherzelle 100 stehenden Wert von a multipliziert. Für die Multiplikation wird der gegebene Programmcode verwendet. Für die Zählung der Wiederholungen wird der in Speicherzelle 96 stehende Wert von n bei jedem Durchlauf um eins verringert. Wiederholt wird, solange dieser Wert noch größer als Null ist. Varianten: Es gibt zahlreiche andere Lösungsmöglichkeiten, beispielsweise unter Verwendung einer Wiederholung mit Anfangsbedingung. Auch die Verwendung symbolischer Adressen ist erlaubt.
3. a)	5	Eine Menge von Prozessen sind in einer Verklemmung, wenn jeder Prozess auf ein Ereignis wartet, das von einem anderen Prozess ausgelöst werden kann. Jedes Fahrzeug wartet wegen der Vorfahrtsregeln auf ein anderes. Fahrzeug A wartet auf Fahrzeug C, Fahrzeug C wartet auf Fahrzeug B, Fahrzeug B wartet auf Fahrzeug A. Durch das gegenseitige Blockieren entsteht eine Verklemmung.

b)	2	Ein Konzept ist das des Monitors. Der Monitor stellt einen geschützten Bereich zur Verfügung, welcher der Kreuzung entspricht und der jeweils nur von einem Fahrzeug genutzt werden kann. Hinweis: Argumentation auch über Semaphore möglich
	40	