Neufassung Juni 2025; verbindlich für die Abiturprüfungen ab 2027

Hinweise zur Verwendung dieses Dokuments

In den folgenden Abschnitten sind Funktionsumfänge und Schreibweisen näher beschrieben, die im Rahmen von zentralen Prüfungsaufgaben im Fach Informatik Verwendung finden. Ein Teil der Abschnitte ist im schriftlichen Abitur als Hilfsmittel für die Prüflinge zugelassen. Diese Abschnitte sind mit einer grün hinterlegten Überschrift versehen. Abschnitte mit orangefarben hinterlegten Überschriften sind nicht als Hilfsmittel zugelassen. Sie sind aber ebenso bedeutsam für zentrale Prüfungsaufgaben. Die als Hilfsmittel zugelassenen Abschnitte werden zudem in einem gesonderten Dokument im Rahmen der fachbezogenen Hinweise für den jeweiligen Abiturjahrgang veröffentlicht.

1 Umgang mit Zeichen und Zeichenketten

Die Prüflinge müssen den Umgang mit den in der folgenden Auflistung genannten „Operationen“ in der im Unterricht verwendeten Programmiersprache beherrschen. Die Verwendung von darüber hinausgehenden Zeichenkettenoperationen ist im Rahmen von zentralen Prüfungsaufgaben nicht zulässig.

- Bestimmen der Länge einer Zeichenkette
- Auslesen eines Zeichens an einer bestimmten Position
- Auslesen einer Teilzeichenkette in einem bestimmten Bereich
- Verbinden von zwei Zeichenketten zu einer
- Prüfen des Inhalts von zwei Zeichenketten auf Gleichheit
- Prüfen einer Zeichenkette auf eine Teilzeichenkette
- Lexikographisches Vergleichen von zwei Zeichenketten
- Bestimmen des ASCII-Werts (Dezimalzahl) zu einem Zeichen (und umgekehrt)

2 Umgang mit mathematischen Operationen

Für die Arbeit mit mathematischen Operationen zur Division und Modulo-Berechnung gelten für programmiersprachenunabhängige Betrachtungen die folgenden Vereinbarungen:

- Der Operator `/` steht bei der Verwendung mit zwei ganzen Zahlen für die ganzzahlige Division ohne Rest, wenn dies in der Aufgabenstellung nicht anders festgelegt ist.
- Der Operator `mod` steht für den (kleinsten, positiven) Rest, der bei der ganzzahligen Division von zwei ganzen Zahlen auftritt.

Bei Aufgabenstellungen zur Implementierung ist von den Prüflingen bei der Verwendung mathematischer Operationen die korrekte Syntax der verwendeten Programmiersprache zu nutzen.

Die Prüflinge müssen die Erzeugung von ganzzahligen Zufallszahlen in einem vorgegebenen Bereich und das ganzzahlige Runden in der im Unterricht verwendeten Programmiersprache beherrschen.

3 Umgang mit statischen Reihungen

In programmiersprachenunabhängigen Betrachtungen beginnt die Nummerierung der Elemente einer statischen Reihung mit dem Index 0.

Bei zweidimensionalen statischen Reihungen erfolgt die Notation der Indizes im Normalfall in der Reihenfolge [Zeile][Spalte], wenn die zugehörigen Daten in einer entsprechenden Tabelle vorliegen. In bestimmten Sachzusammenhängen ist auch eine andere Notation möglich. Dies ist in zentralen Prüfungsaufgaben dann deutlich beschrieben.

Die Prüflinge müssen in der verwendeten Programmiersprache die Größe von ein- und zweidimensionalen statischen Reihungen auslesen können.

Statische Reihungen sind iterierbar, so dass die Verwendung einer for-each-Schleife zum elementweisen Auslesen einer statischen Reihung möglich ist.

4 Operationen für dynamische Reihungen, Stapel, Schlangen und Binärbäume

Im Rahmen von zentralen Prüfungsaufgaben werden für die aufgeführten Klassen die folgenden Operationen verwendet. Die Klassen verwenden Inhaltstypen (bzw. Inhaltsklassen), die jeweils der aktuellen Aufgabenstellung angepasst werden. Mögliche Laufzeitfehler bei der Anwendung der Operationen, z. B. Entnehmen bei einem leeren Stapel, Löschen oder Hinzufügen eines Elements an einer nicht existierenden Position einer dynamischen Reihung oder Auslesen des Wurzelelements bei einem leeren Baum, müssen bei der Verwendung durch entsprechende Abfragen explizit ausgeschlossen werden.

Dynamische Reihung

Die Nummerierung der Elemente der dynamischen Reihung beginnt mit dem Index 0. Dynamische Reihungen sind iterierbar, so dass die Verwendung einer for-each-Schleife zum elementweisen Auslesen einer dynamischen Reihung möglich ist.

`DynArray()`
Eine leere dynamische Reihung wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die dynamische Reihung kein Element enthält, wird der Wert `wahr` zurückgegeben, sonst der Wert `falsch`.

`getItem(index: Ganzzahl): Inhaltstyp`
Der Inhalt des Elements an der Position `index` wird zurückgegeben.

`append(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird am Ende der dynamischen Reihung angefügt.

`insertAt(index: Ganzzahl, inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird an der Position `index` in die dynamische Reihung eingefügt. Das Element, das sich vorher an dieser Position befunden hat, und alle nachfolgenden werden nach hinten verschoben. Entspricht der Wert von `index` der Länge der dynamischen Reihung, so wird ein neues Element am Ende der dynamischen Reihung angefügt.

`setItem(index: Ganzzahl, inhalt: Inhaltstyp)`
Der Inhalt des Elementes an der Position `index` wird durch den übergebenen Inhalt ersetzt.

`delete(index: Ganzzahl)`
Das Element an der Position `index` wird entfernt. Alle folgenden Elemente werden um eine Position nach vorne geschoben.

`getLength(): Ganzzahl`
Die Anzahl der Elemente der dynamischen Reihung wird zurückgegeben.

Stapel

`Stack()`
Ein leerer Stapel wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn der Stapel kein Element enthält, wird der Wert `wahr` zurückgegeben, sonst der Wert `falsch`.

`top(): Inhaltstyp`
Der Inhalt des obersten Elements des Stapels wird zurückgegeben, das Element aber nicht entnommen.

`push(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird oben auf den Stapel gelegt.

`pop(): Inhaltstyp`
Das oberste Element des Stapels wird entnommen. Der Inhalt dieses Elements wird zurückgegeben.

Schlange

`Queue()`
Eine leere Schlange wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die Schlange kein Element enthält, wird der Wert `wahr` zurückgegeben, sonst der Wert `falsch`.

`head(): Inhaltstyp`
Der Inhalt des vordersten Elements der Schlange wird zurückgegeben, das Element aber nicht entnommen.

`enqueue(inhalt: Inhaltstyp)`
Ein neues Element mit dem angegebenen Inhalt wird am Ende an die Schlange angehängt.

`dequeue(): Inhaltstyp`
Das vorderste Element der Schlange wird entnommen. Der Inhalt dieses Elements wird zurückgegeben.

Binärbaum

`BinTree()`
Ein leerer Baum wird erzeugt. Er besitzt keinen Inhalt und keine Teilbäume.

`BinTree(inhalt: Inhaltstyp)`
Ein Baum wird erzeugt. Die Wurzel erhält den übergebenen Inhalt als Wert. Der Baum besitzt jeweils einen leeren Baum als linken und rechten Teilbaum.

`isEmpty(): Wahrheitswert`
Wenn der Baum ein leerer Baum ist, wird der Wert `wahr` zurückgegeben, sonst der Wert `falsch`.

`getItem(): Inhaltstyp`
Die Operation gibt den Inhaltswert der Wurzel des Baumes zurück.

`setItem(inhalt: Inhaltstyp)`
Die Wurzel des Baums erhält den übergebenen Inhalt als Wert.
Bei einem leeren Baum wird zusätzlich als linker und rechter Teilbaum jeweils ein leerer Baum gesetzt.

`isLeaf(): Wahrheitswert`
Wenn der Baum jeweils einen leeren Baum als linken und rechten Teilbaum besitzt, also ein Blatt ist, wird der Wert `wahr` zurückgegeben, sonst der Wert `falsch`.

`getLeft(): Binärbaum`
Die Operation gibt den linken Teilbaum zurück.

`setLeft(b: Binärbaum)`
Der übergebene Baum wird als linker Teilbaum gesetzt.

`getRight(): Binärbaum`
Die Operation gibt den rechten Teilbaum zurück.

`setRight(b: Binärbaum)`
Der übergebene Baum wird als rechter Teilbaum gesetzt.

`setEmpty()`
Der Baum wird zu einem leeren Baum, d. h. er besitzt keinen Inhalt und keine Teilbäume.

5 Standardisierte Darstellung von Operationen und Algorithmen

Die Signaturen von Operationen werden in den Aufgabenstellungen vergleichbar zur Darstellung in Klassendiagrammen wie folgt notiert, um die Bezeichnung, die Übergabeparameter und den Rückgabewert zu kennzeichnen:

```
Bezeichnung(Parameterbezeichnung: Parametertyp, ...): Rückgabetyp
```

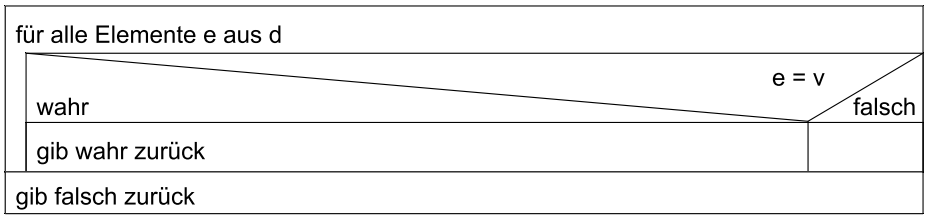
Beispiele:
```
istGerade(x: Ganzzahl): Wahrheitswert
minimum(a, b: Fließkommazahl): Fließkommazahl
sortiere(a: dynamische Reihung vom Inhaltstyp Zeichenkette)
```

In programmiersprachenunabhängigen Betrachtungen beginnt die Nummerierung der Positionen in einer Zeichenkette sowie bei statischen und dynamischen Reihungen mit 0 (nullbasierte Indizierung). Bei Aufgabenstellungen zur Implementierung ist von den Prüflingen die gängige Indizierung der verwendeten Programmiersprache zu nutzen.

Zur Darstellung von Algorithmen werden in den zentralen Prüfungsaufgaben Struktogramme verwendet, wie in den folgenden Beispielen exemplarisch dargestellt. Die Formulierungen im Struktogramm sind unabhängig von einer verwendeten Programmiersprache zu wählen. Kommentare im Struktogramm werden durch zwei vorangestellte Schrägstriche dargestellt. Erfolgt im Programmablauf eine Rückgabe („gib … zurück“), so wird die Operation an dieser Stelle beendet.

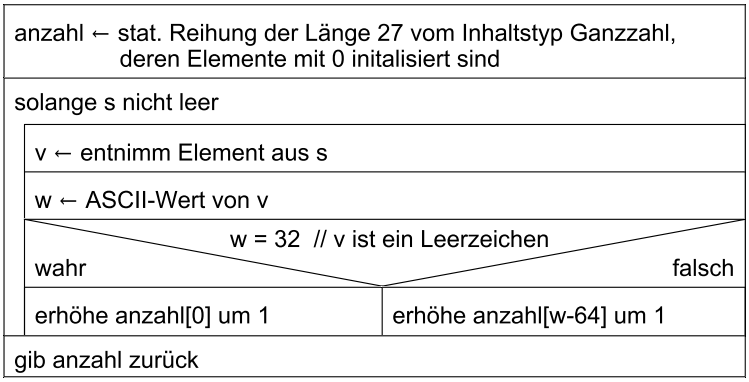
Beispiele für Struktogramme, die unterschiedliche Notationsformen verdeutlichen:

istEnthalten(d: dyn. Reihung vom Inhaltstyp Ganzzahl, v: Ganzzahl): Wahrheitswert



Beispiel 1: Der Algorithmus nutzt eine for-each-Schleife zum vollständigen Durchlaufen der dynamischen Reihung

anzahlZeichen(s: Stapel vom Inhaltstyp Zeichen):
stat. Reihung vom Inhaltstyp Ganzzahl



Beispiel 2: Der Algorithmus verarbeitet einen Stapel, von dem bekannt ist, dass dieser nur Großbuchstaben (A-Z) und Leerzeichen enthält. Die Notation von Anweisungen erfolgt hier sprachlich, aber auch formalere Schreibweisen wie „anzahl ← anzahl + 1“ sind möglich.

erzeugePixel(): dynamische Reihung vom Inhaltstyp Pixel

f ← leere dynamische Reihung vom Inhaltstyp Pixel

solange f nicht leer

lösche Element an Position 0 von f

anzahlRot ← 0

wiederhole 100-mal

z ← ganzzahlige Zufallszahl zwischen 1 und 4

z ist gleich: 1	2	3	4	sonst
p ← erzeuge Pixel(255,0,0)	p ← erzeuge Pixel(0,255,0)	p ← erzeuge Pixel(0,0,255)	p ← erzeuge Pixel(120,75,250)	
anzahlRot ← anzahlRot + 1				

füge p zu f hinzu

wiederhole bis anzahlRot > 30

gib f zurück

Beispiel 3: Durch den Befehl „erzeuge …“ wird ein Objekt des angegebenen Typs über die entsprechende Konstruktoroperation erzeugt.

istEnthalten(b: Binärbaum vom Inhaltstyp Person, n: Zeichenkette): Wahrheitswert

b.isEmpty(): wahr	falsch
gib falsch zurück	

aktuell ← b.getItem()

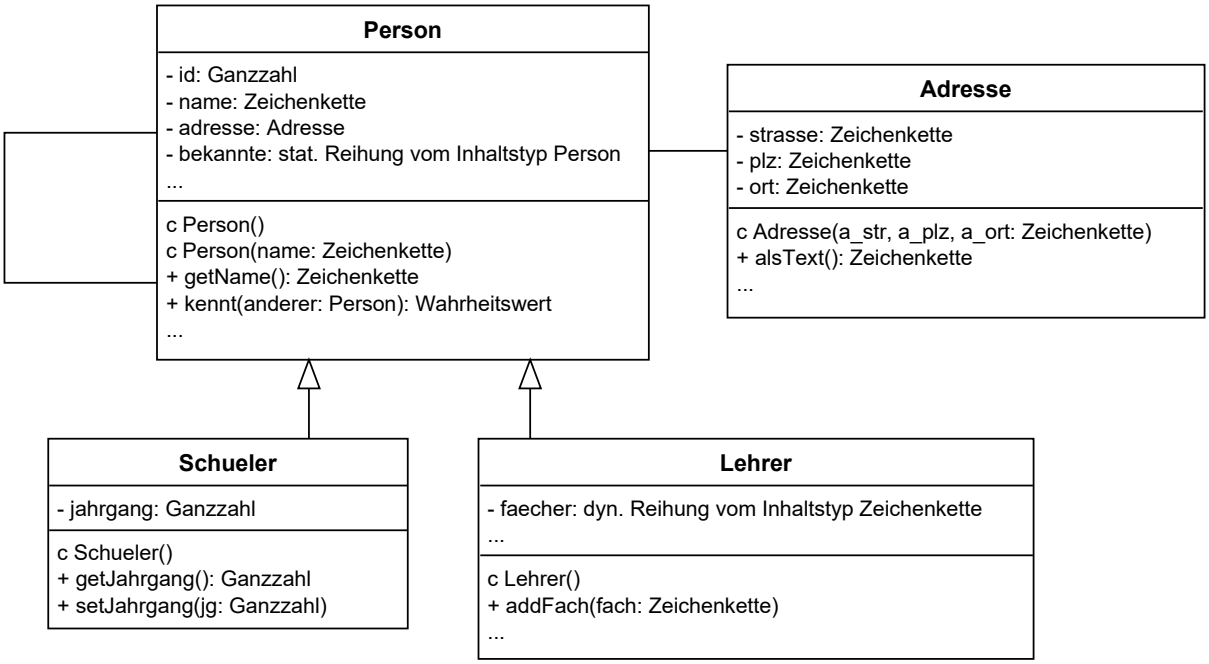
aktuell.getNachname() = n: wahr	falsch
gib wahr zurück	links ← istEnthalten(b.getLeft(), n)
	rechts ← istEnthalten(b.getRight(), n)

links = wahr ODER rechts = wahr: wahr	falsch
gib wahr zurück	gib falsch zurück

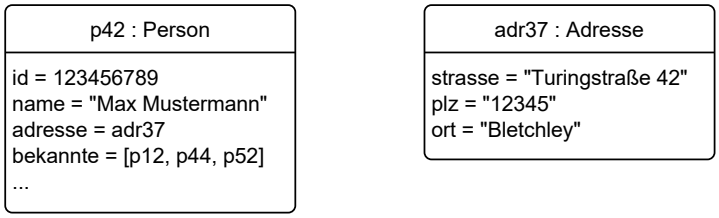
Beispiel 4: Der Algorithmus arbeitet rekursiv. Die Notation von Zugriffsoperationen erfolgt hier durch direkte Verwendung der Operationen der Klasse BinTree (z. B. `b.isEmpty()` statt „b ist leer“).

6 Notation von Klassen- und Objektdiagrammen

Klassendiagramme werden in den zentralen Prüfungsaufgaben wie im folgenden Beispiel exemplarisch notiert. Im Sinne einer übersichtlicheren Darstellung können vereinfacht nur die für die konkrete Aufgabenstellung wichtigen Attribute und Operationen angegeben werden, wie im Beispiel in der Klasse `Lehrer` dokumentiert. Nicht aufgeführte Attribute bzw. Operationen werden durch eine Pünktchen-Schreibweise verdeutlicht. Konstruktor-Operationen werden durch ein vorangestelltes `c` notiert. Assoziationen zwischen Klassen werden stets unspezifiziert durch einfache Verbindungslinien angegeben. Die Kennzeichnung der Sichtbarkeit (+ und -) bei Attributen und Operationen ist im Normalfall obligatorisch.



Eine Objektkarte dokumentiert die konkrete Belegung der Attribute für ein konkretes Objekt einer Klasse. Da die Anzahl der Attribute sehr groß sein kann, werden ggf. nur die Attribute aufgelistet, die für die Aufgabenstellung bedeutsam sind. Die Notation eines Objektdiagramms orientiert sich an den Vorgaben für die Klassendiagramme.

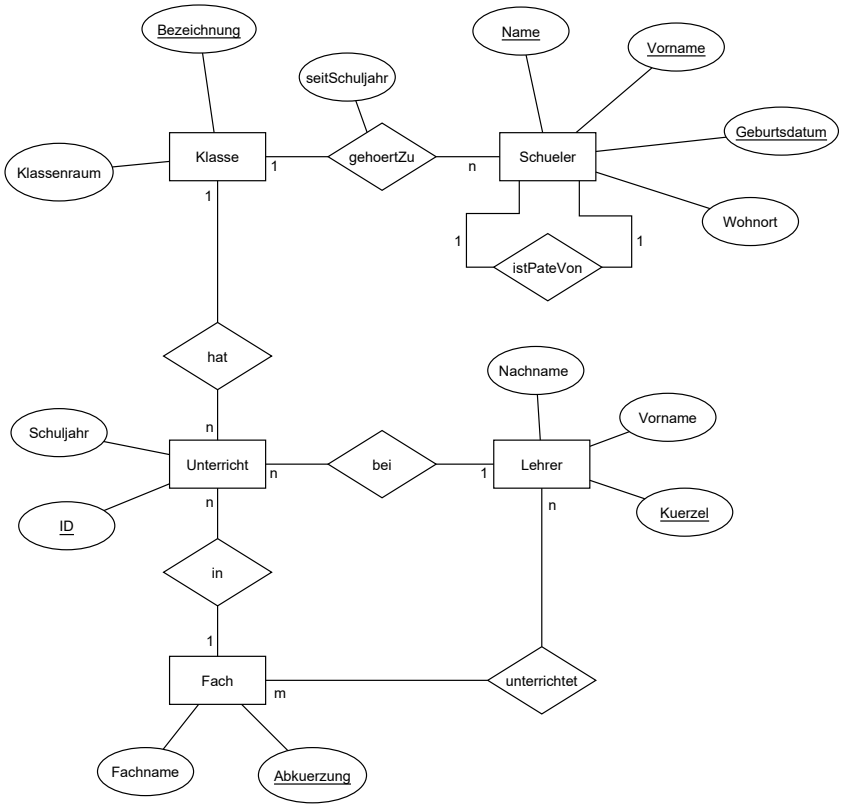


7 Kurzschreibweise von Datenbanktabellen / Notation von ER-Diagrammen

Bei der Darstellung eines relationalen Datenbankschemas in Kurzschreibweise werden die Primärschlüssel durch Unterstreichen und die Fremdschlüssel durch Voranstellen eines Pfeils gekennzeichnet, wie im folgenden Beispiel dokumentiert:

Produkt(Nummer, Bezeichnung, Anzahl, ↑LKuerzel)
Lieferant(Kuerzel, Firmenname, Ort)
Bestellung(↑PNummer, ↑LKuerzel, Preis)

Im folgenden Beispiel sind die Komponenten eines ER-Diagramms dargestellt, wie sie in den zentralen Prüfungsaufgaben verwendet werden. Attribute können sowohl Entitätstypen als auch Beziehungstypen zugeordnet werden. Auch rekursive Beziehungen sind möglich. Die Darstellung der Kardinalitäten an den Kanten und die Kennzeichnung von Schlüsselattributen der Entitätstypen sind im Normalfall obligatorisch.



8 Datenbankabfragen

In den Aufgabenstellungen und in den erwarteten Lösungen wird die Abfragesprache SQL mit dem folgenden Sprachumfang unter Berücksichtigung der angegebenen Operatoren und Aggregatfunktionen verwendet. Verschachtelte SQL-Anweisungen werden im Rahmen der zentralen Prüfungsaufgaben nicht thematisiert.

SELECT-Anweisung:

```
SELECT [DISTINCT] * | spalte1 [AS s1], spalte2 [AS s2], …
FROM tabelle1 [t1], tabelle2 [t2], …
[WHERE bedingung1 (AND | OR) bedingung2 (AND | OR) … ]
[GROUP BY spalte1, spalte2, …
[HAVING gruppenBedingung1 (AND | OR) gruppenBedingung2 (AND | OR) …]]
[ORDER BY spalte1 [ASC | DESC], spalte2 [ASC | DESC], …]
[LIMIT anzahl]
```

Angaben in eckigen Klammern sind optional. Spalten können Attribute, Aggregatfunktionen oder Berechnungen sein. Bei `GROUP BY` und `ORDER BY` ist auch die Angabe eines Alias möglich.

Operatoren für Berechnungen: +, –, *, /

Operatoren für Vergleiche in Bedingungen: =, != (ungleich), >, <, >=, <=, `NOT`, `IS NULL`, `IN`, `BETWEEN`, `LIKE` (mit den Platzhaltern _ (für genau ein Zeichen) und `%` (für beliebig viele Zeichen))

Aggregatfunktionen: `AVG`, `COUNT`, `MAX`, `MIN`, `SUM`

Hinweis zur Verwendung von `GROUP BY`: Bei einer Gruppierung müssen alle Spalten in der SELECT-Klausel entweder von einer Aggregatfunktion umschlossen oder explizit in der GROUP BY-Klausel aufgezählt sein.

9 Notation von Lauflängencodierungen / Berechnung von Kompressionsraten

Lauflängencodierung

Bei einer Lauflängencodierung werden Sequenzen von identischen Werten durch Repräsentanten der Anzahl und des Werts ersetzt. Die Notation erfolgt in der Reihenfolge Anzahl-Wert.
Aus der Folge `rrrssssgrr` wird so die codierte Folge `3r4s1g2r`. Vorgaben für die Notation der Anzahlen, z. B. nur einstellige Zahlen, ergeben sich aus dem jeweiligen Aufgabenkontext.

Im Rahmen von zentralen Prüfungsaufgaben ist es nicht notwendig, die lauflängencodierte Folge in Binärcode-Schreibweise zu notieren, wenn dies nicht explizit verlangt ist.

Kompressionsverhältnis und prozentuale Datenersparnis

Das *Kompressionsverhältnis (compression ratio)* ist das Verhältnis der Größe der komprimierten Daten zur Größe der Originaldaten:
Kompressionsverhältnis = Größe der komprimierten Daten / Größe der Originaldaten

Die *prozentuale Datenersparnis* gibt an, wie viel Prozent an Speicherplatz durch die Kompression eingespart wurde. Sie wird wie folgt berechnet:
Prozentuale Datenersparnis = (1 – Größe der komprimierten Daten / Größe der Originaldaten) · 100 %

10 Notation von Bitfolgen nach dem (7,4)-Hamming-Code

Die Daten- und Prüfbits im (7,4)-Hamming-Code werden in der Reihenfolge `p0 p1 d0 p2 d1 d2 d3` notiert.

Die Kontrollgruppen sind entsprechend der folgenden Abbildung zusammengesetzt:

	Datenbits			
Prüfbit	d0	d1	d2	d3
p0	x	x	-	x
p1	x	-	x	x
p2	-	x	x	x

Die Festlegung der Prüfbits erfolgt auf Basis einer geraden Parität.

11 ASCII-Tabelle / Vigenère-Verschlüsselungs-Quadrat / Alphabet mit Nummerierung

Auszug aus der ASCII-Tabelle

dez	char	dez	char	dez	char	dez	char	dez	char	dez	char
…	…	48	0	65	A	82	R	99	c	116	t
32	Leerz.	49	1	66	B	83	S	100	d	117	u
33	!	50	2	67	C	84	T	101	e	118	v
34	"	51	3	68	D	85	U	102	f	119	w
35	#	52	4	69	E	86	V	103	g	120	x
36	$	53	5	70	F	87	W	104	h	121	y
37	%	54	6	71	G	88	X	105	i	122	z
38	&	55	7	72	H	89	Y	106	j	123	{
39	'	56	8	73	I	90	Z	107	k	124	|
40	(	57	9	74	J	91	[	108	l	125	}
41	)	58	:	75	K	92	\	109	m	126	~
42	*	59	;	76	L	93	]	110	n	…	…
43	+	60	<	77	M	94	^	111	o		
44	,	61	=	78	N	95	_	112	p		
45	-	62	>	79	O	96	`	113	q		
46	.	63	?	80	P	97	a	114	r		
47	/	64	@	81	Q	98	b	115	s		

Vigenère-Verschlüsselungs-Quadrat

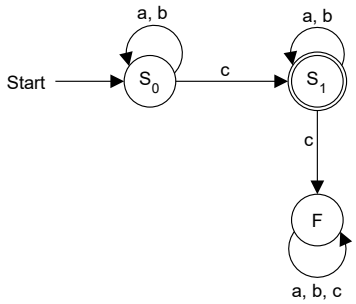
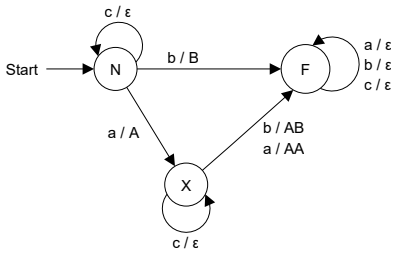
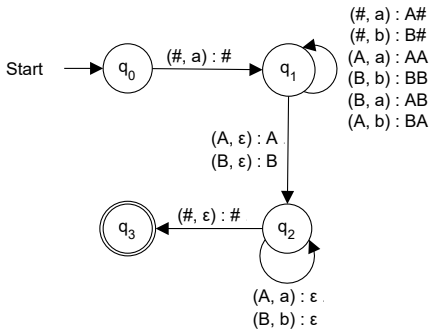
Schlüsselbuchstabe \ Klartextbuchstabe	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Alphabet mit Nummerierung

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

12 Notation von endlichen Automaten, von Mealy-Automaten und von Kellerautomaten

Im Rahmen von zentralen Prüfungsaufgaben können auf grundlegendem Anforderungsniveau deterministische endliche Automaten (DEA) und Mealy-Automaten thematisiert werden. Kellerautomaten (NKA) sind nur Bestandteil von Prüfungsaufgaben auf erhöhtem Anforderungsniveau.

Ein deterministischer endlicher Automat (DEA) wird durch die Angabe eines Eingabealphabets Σ und durch einen Zustandsgraphen dargestellt. Die grafische Darstellung entspricht dem nebenstehenden Beispiel. Endliche Automaten werden, wenn nicht explizit anders vermerkt, vollständig angegeben. Auf die Darstellung eines Fehlerzustands im Zustandsgraphen kann verzichtet werden, wenn dies durch eine Erläuterung kenntlich gemacht wird.	Eingabealphabet Σ = {a, b, c} 
Ein Mealy-Automat wird durch die Angabe eines Eingabealphabets Σ, eines Ausgabealphabets Ω und durch einen vollständigen Übergangsgraphen dargestellt. Die Übergänge eines Mealy-Automaten werden wie nebenstehend dargestellt. Am Zustandsübergang erfolgt die Schreibweise in der Reihenfolge Eingabezeichen / Ausgabezeichen. Zur Vereinfachung sind an den Zustandsübergängen statt eines einzelnen Ausgabezeichens auch Wörter über dem Ausgabealphabet erlaubt. Es sind auch Übergänge möglich, bei denen keine Ausgabe erfolgt. Eine leere Ausgabe wird mit dem Zeichen ε gekennzeichnet.	Eingabealphabet Σ = {a, b, c} Ausgabealphabet Ω = {A, B} 
Bei nichtdeterministischen Kellerautomaten (NKA) wird am Zustandsübergang in Klammern zuerst das oberste Zeichen des Kellerspeichers gefolgt vom aktuellen Eingabezeichen notiert. Das vom Speicher gelesene Zeichen wird dabei aus dem Keller entfernt. Hinter dem Doppelpunkt wird notiert, welches Zeichen wieder im Kellerspeicher abgelegt wird. Dabei ist auch das Speichern mehrerer Zeichen möglich. Das am weitesten rechts notierte Zeichen wird dann zuerst auf den Keller gelegt. Der Kellerspeicher besitzt das Vorbelegungszeichen #. Es sind ε-Übergänge möglich. Eine Eingabe wird genau dann akzeptiert, wenn es eine Möglichkeit gibt, so dass sich der Automat nach vollständiger Verarbeitung der Eingabe in einem Endzustand befindet.	Eingabealphabet Σ = {a, b, c} Kelleralphabet Γ = {A, B, #} 

13 Notation von Grammatiken formaler Sprachen

Eine Grammatik einer formalen Sprache wird durch die Angabe der Menge der Nichtterminalsymbole, der Menge der Terminalsymbole, das Startsymbol und die Produktionsregeln angegeben.

Die Nichtterminalsymbole unterscheiden sich bzgl. der Notation in der Regel deutlich von den Terminalsymbolen, z. B. durch die Verwendung von Großbuchstaben und von Kleinbuchstaben.

Die Verwendung von ε als leerem Zeichen ist zulässig.

Ist in Prüfungsaufgaben allgemein von einer Grammatik die Rede, so ist eine beliebige kontextfreie Grammatik gemeint, wenn nicht explizit eine reguläre Grammatik thematisiert wird. Reguläre Grammatiken werden im Rahmen von Prüfungsaufgaben in der Regel rechtsregulär notiert, wie im nebenstehenden Beispiel dokumentiert.

```
N = {A, B, S}
T = {1, 2, a, c}
```

Startsymbol: `S`

Produktionsregeln:

```
S → 1A | 2B
A → 1B
B → aS | cS | a | c
```