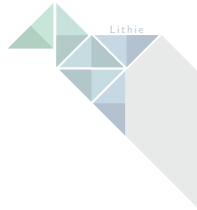


# Mathe. Haha. Kann ich.

Tutorium Unostasio

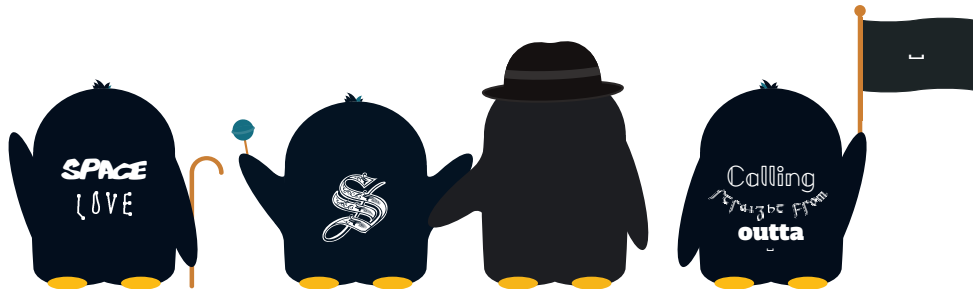
Florian Sihler ◦ KW 44

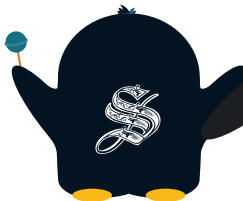
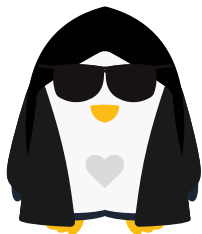






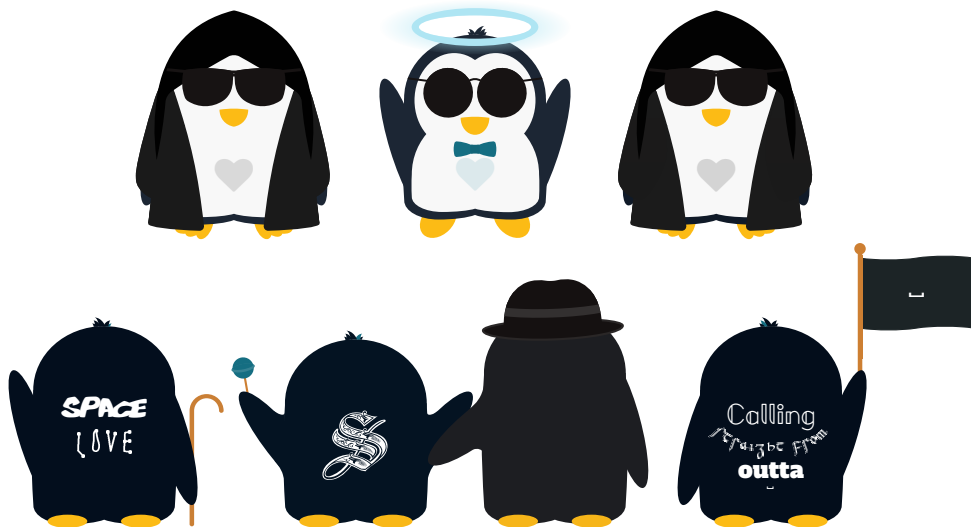








All hail the space!





All hail the space!  
Lehrfeld? Lärfeld? Learfeld? Nix?



# Präsenzaufgabe

1

Kommen Sie hier rein?

# Präsenzaufgabe

1

Kommen Sie hier rein?

Konstruieren Sie für jede der folgenden Aussagen einen booleschen Ausdruck, welcher diese überprüft.

# Präsenzaufgabe

1

Kommen Sie hier rein?

Konstruieren Sie für jede der folgenden Aussagen einen booleschen Ausdruck, welcher diese überprüft.

1. Eine Person ist Teenager. (`int` `alter` in Jahren)

# Präsenzaufgabe

1

Kommen Sie hier rein?

Konstruieren Sie für jede der folgenden Aussagen einen booleschen Ausdruck, welcher diese überprüft.

1. Eine Person ist Teenager. (`int` `alter` in Jahren)
2. Es ist Vormittag. (`int` `uhrzeit` im 24h-Format)

# Präsenzaufgabe

1

Kommen Sie hier rein?

Konstruieren Sie für jede der folgenden Aussagen einen booleschen Ausdruck, welcher diese überprüft.

1. Eine Person ist Teenager. (`int` `alter` in Jahren)
2. Es ist Vormittag. (`int` `uhrzeit` im 24h-Format)
3. Eine Tür ist geschlossen. (`boolean` `istOffen`)

# Präsenzaufgabe

1

Kommen Sie hier rein?

Konstruieren Sie für jede der folgenden Aussagen einen booleschen Ausdruck, welcher diese überprüft.

1. Eine Person ist Teenager. (`int` alter in Jahren)
2. Es ist Vormittag. (`int` uhrzeit im 24h-Format)
3. Eine Tür ist geschlossen. (`boolean` istOffen)
4. Im Kaffee ist entweder Zucker oder Milch, aber nicht beides.  
(`boolean` zucker und `boolean` milch)

# Präsenzaufgabe - Lösung



# Präsenzaufgabe - Lösung

- Teenager:

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag:

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür:

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee:

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch)`



# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:`

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

*^ entspricht dem XOR-Operator.*

A	B	A ^ B

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

*^ entspricht dem XOR-Operator.*

A	B	A ^ B
0	0	0

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

*^ entspricht dem XOR-Operator.*

A	B	A ^ B
0	0	0
0	1	1

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

*^ entspricht dem XOR-Operator.*

A	B	A ^ B
0	0	0
0	1	1
1	0	1

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

*^ entspricht dem XOR-Operator.*

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

*^ entspricht dem XOR-Operator. Dieser evaluiert nur genau dann zu wahr,*

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0



# Präsenzaufgabe - Lösung

- Teenager: `alter >= 13 && alter <= 18` (oooder 19?)
- Vormittag: `uhrzeit >= 9 && uhrzeit <= 12`
- Tür: `!istOffen` (But who trusts the name?)
- (Guter?) Kaffee: `(zucker || milch) && !(zucker && milch), oder:  
zucker ^ milch.`

*^ entspricht dem XOR-Operator. Dieser evaluiert nur genau dann zu wahr, wenn einer der Parameter wahr und der andere falsch ist.*

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

# Übungsblatt 1 - Aufgabe 1 a)

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1$$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1$$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)}$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1$



# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)}$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5)$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$
- Hexadezimal:

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$
- Hexadezimal:

$$1_{(16)} \hat{=} 1_{(10)}$$

$$2_{(16)} \hat{=} 2_{(10)}$$

$$3_{(16)} \hat{=} 3_{(10)}$$

$$4_{(16)} \hat{=} 4_{(10)}$$

$$5_{(16)} \hat{=} 5_{(10)}$$

$$6_{(16)} \hat{=} 6_{(10)}$$

$$7_{(16)} \hat{=} 7_{(10)}$$

$$8_{(16)} \hat{=} 8_{(10)}$$

$$9_{(16)} \hat{=} 9_{(10)}$$

$$A_{(16)} \hat{=} 10_{(10)}$$

$$B_{(16)} \hat{=} 11_{(10)}$$

$$C_{(16)} \hat{=} 12_{(10)}$$

$$D_{(16)} \hat{=} 13_{(10)}$$

$$E_{(16)} \hat{=} 14_{(10)}$$

$$F_{(16)} \hat{=} 15_{(10)}$$

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$

- Hexadezimal:

$$1_{(16)} \hat{=} 1_{(10)}$$

$$2_{(16)} \hat{=} 2_{(10)}$$

$$3_{(16)} \hat{=} 3_{(10)}$$

$$4_{(16)} \hat{=} 4_{(10)}$$

$$5_{(16)} \hat{=} 5_{(10)}$$

$$6_{(16)} \hat{=} 6_{(10)}$$

$$7_{(16)} \hat{=} 7_{(10)}$$

$$8_{(16)} \hat{=} 8_{(10)}$$

$$9_{(16)} \hat{=} 9_{(10)}$$

$$A_{(16)} \hat{=} 10_{(10)}$$

$$B_{(16)} \hat{=} 11_{(10)}$$

$$C_{(16)} \hat{=} 12_{(10)}$$

$$D_{(16)} \hat{=} 13_{(10)}$$

$$E_{(16)} \hat{=} 14_{(10)}$$

$$F_{(16)} \hat{=} 15_{(10)}$$

- FACE<sub>(16)</sub>

# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$

- Hexadezimal:

$$1_{(16)} \hat{=} 1_{(10)}$$

$$2_{(16)} \hat{=} 2_{(10)}$$

$$3_{(16)} \hat{=} 3_{(10)}$$

$$4_{(16)} \hat{=} 4_{(10)}$$

$$5_{(16)} \hat{=} 5_{(10)}$$

$$6_{(16)} \hat{=} 6_{(10)}$$

$$7_{(16)} \hat{=} 7_{(10)}$$

$$8_{(16)} \hat{=} 8_{(10)}$$

$$9_{(16)} \hat{=} 9_{(10)}$$

$$A_{(16)} \hat{=} 10_{(10)}$$

$$B_{(16)} \hat{=} 11_{(10)}$$

$$C_{(16)} \hat{=} 12_{(10)}$$

$$D_{(16)} \hat{=} 13_{(10)}$$

$$E_{(16)} \hat{=} 14_{(10)}$$

$$F_{(16)} \hat{=} 15_{(10)}$$

- $FACE_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14$



# Übungsblatt 1 - Aufgabe 1 a)

- Hornerschema am Beispiel von  $1101_{(2)}$ :

$$(((1 \cdot 2 + 1) \cdot 2 + 0) \cdot 2) + 1 = (((3) \cdot 2 + 0) \cdot 2) + 1 = (6 \cdot 2) + 1 = 13_{(10)}.$$

- $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$

- Hexadezimal:

$$1_{(16)} \hat{=} 1_{(10)}$$

$$2_{(16)} \hat{=} 2_{(10)}$$

$$3_{(16)} \hat{=} 3_{(10)}$$

$$4_{(16)} \hat{=} 4_{(10)}$$

$$5_{(16)} \hat{=} 5_{(10)}$$

$$6_{(16)} \hat{=} 6_{(10)}$$

$$7_{(16)} \hat{=} 7_{(10)}$$

$$8_{(16)} \hat{=} 8_{(10)}$$

$$9_{(16)} \hat{=} 9_{(10)}$$

$$A_{(16)} \hat{=} 10_{(10)}$$

$$B_{(16)} \hat{=} 11_{(10)}$$

$$C_{(16)} \hat{=} 12_{(10)}$$

$$D_{(16)} \hat{=} 13_{(10)}$$

$$E_{(16)} \hat{=} 14_{(10)}$$

$$F_{(16)} \hat{=} 15_{(10)}$$

- $FACE_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}$

# Übungsblatt 1 - Aufgabe 1 a)

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen!

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}.$

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}.$

1

+

=

1

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = ((((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1 = 85_{(10)}.$

$$\begin{array}{rcl} 1 & & 0 \\ + & & + \\ & & 2 \\ = & \nearrow \cdot 2 & = \\ 1 & & 2 \end{array}$$

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}.$

1	0	1
+	+	+
	2	4
=	=	=
1	2	5

Diagram illustrating the iterative calculation of the decimal value of the binary number 1010101. The diagram shows three columns of numbers. The first column contains 1, +, and =. The second column contains 0, +, 2, =, and 2. The third column contains 1, +, 4, =, and 5. Arrows indicate the sequence of operations: from the first column's 1 to the second column's 2, and from the second column's 2 to the third column's 5. The arrows are labeled with ".2", representing multiplication by 2.



# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}.$

1	0	1	0
+	+	+	+
	2	4	10
=	=	=	=
1	2	5	10

Diagram illustrating the step-by-step calculation of the decimal value of the binary number 1010101. The diagram shows the sequence of operations: starting with 1, then multiplying by 2 and adding 0 to get 2, then multiplying by 2 and adding 1 to get 5, then multiplying by 2 and adding 0 to get 10, and finally multiplying by 2 and adding 1 to get 21. The diagram uses arrows and the notation  $\cdot 2$  to indicate the multiplication step, and  $+$  to indicate the addition step.

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}.$

1	0	1	0	1
+	+	+	+	+
=	2	4	10	20
1	2	5	10	21

Diagram illustrating the step-by-step calculation of the decimal value of the binary number 1010101. The diagram shows the sequence of operations: starting with 1, then adding 0 (result 2), adding 1 (result 4), adding 0 (result 10), adding 1 (result 20), and finally adding 1 (result 21). The operations are indicated by arrows labeled ".2" (representing multiplication by 2) and "+" (representing addition of the next bit).

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}.$

1	0	1	0	1	0
+	+	+	+	+	+
=	2	4	10	20	42
1	2	5	10	21	42

Diagram illustrating the step-by-step calculation of the decimal value of the binary number 1010101. The diagram shows the sequence of operations: starting with 1, then multiplying by 2 and adding the next bit (0) to get 2, then multiplying by 2 and adding the next bit (1) to get 4, then multiplying by 2 and adding the next bit (0) to get 10, then multiplying by 2 and adding the next bit (1) to get 20, then multiplying by 2 and adding the next bit (0) to get 42, and finally multiplying by 2 and adding the last bit (1) to get 85. The diagram uses arrows to show the progression of the calculation.

# Übungsblatt 1 - Aufgabe 1 a)

- Wir können die Rechnungen auch tabellarisch veranschaulichen! Zur Erinnerung:  
 $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}.$

1	0	1	0	1	0	1
+	+	+	+	+	+	+
	2	4	10	20	42	84
=	=	=	=	=	=	=
1	2	5	10	21	42	85

Diagram illustrating the step-by-step calculation of the decimal value of the binary number 1010101. The diagram shows the sequence of operations: starting with 1, then multiplying by 2 and adding the next bit (0) to get 2, then multiplying by 2 and adding the next bit (1) to get 5, and so on, resulting in the final value 85.

# Übungsblatt 1 - Aufgabe 1 a)

# Übungsblatt 1 - Aufgabe 1 a)

- Analog:

# Übungsblatt 1 - Aufgabe 1 a)

- Analog:  $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}.$

# Übungsblatt 1 - Aufgabe 1 a)

- Analog:  $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}.$

1

+

=

1



# Übungsblatt 1 - Aufgabe 1 a)

- Analog:  $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}.$

$$\begin{array}{rcl} 1 & & 2 \\ + & & + \\ = & \nearrow \cdot 8 & = \\ 1 & & 10 \end{array}$$

# Übungsblatt 1 - Aufgabe 1 a)

- Analog:  $12345_{(8)} = (((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5 = 5349_{(10)}$ .

1	2	3
+	+	+
	8	80
=	=	=
1	10	83

The diagram illustrates the iterative conversion of the base-8 number 12345 to base-10. It shows three stages of the calculation:

- Stage 1:  $1 \cdot 8 + 2 = 10$ . An arrow labeled  $\cdot 8$  points from the '1' to the '8'.
- Stage 2:  $10 \cdot 8 + 3 = 83$ . An arrow labeled  $\cdot 8$  points from the '10' to the '80'.

# Übungsblatt 1 - Aufgabe 1 a)

- Analog:  $12345_{(8)} = (((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5 = 5349_{(10)}$ .

1	2	3	4
+	+	+	+
	8	80	664
=	=	=	=
1	10	83	668

Diagram illustrating the conversion of the base-8 number 12345 to base-10. The calculation is shown in four columns, corresponding to the digits 1, 2, 3, and 4. Each column shows the digit, a plus sign, the result of the previous step multiplied by 8, an equals sign, and the final result. Arrows indicate the flow of the calculation from left to right.

# Übungsblatt 1 - Aufgabe 1 a)

- Analog:  $12345_{(8)} = (((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}.$

1	2	3	4	5
+	+	+	+	+
	8	80	664	5344
=	=	=	=	=
1	10	83	668	5349

Diagram illustrating the conversion of the base-8 number 12345 to the base-10 number 5349 using the Horner's method. The diagram shows the intermediate results of the calculation:

- 1 (base 8) is converted to 1 (base 10).
- 1 \* 8 + 2 = 10 (base 10).
- 10 \* 8 + 3 = 83 (base 10).
- 83 \* 8 + 4 = 668 (base 10).
- 668 \* 8 + 5 = 5349 (base 10).

# Übungsblatt 1 - Aufgabe 1 a)

# Übungsblatt 1 - Aufgabe 1 a)

- Sowie:

# Übungsblatt 1 - Aufgabe 1 a)

- Sowie:  $\text{FACE}_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}.$

# Übungsblatt 1 - Aufgabe 1 a)

- Sowie:  $\text{FACE}_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}$ .

F  
+  
  
=  
15



# Übungsblatt 1 - Aufgabe 1 a)

- Sowie:  $\text{FACE}_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}$ .

$$\begin{array}{rcl} \text{F} & & \text{A} \\ + & & + \\ & & 240 \\ = & \nearrow \cdot 16 & = \\ 15 & & 250 \end{array}$$

# Übungsblatt 1 - Aufgabe 1 a)

- Sowie:  $\text{FACE}_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}$ .

F	A	C
+	+	+
	240	4000
=	=	=
15	250	4012

Diagram illustrating the conversion of hexadecimal digits to decimal values:

- $15 \cdot 16 = 240$  (indicated by an arrow from 15 to 240)
- $250 \cdot 16 = 4000$  (indicated by an arrow from 250 to 4000)

# Übungsblatt 1 - Aufgabe 1 a)

- Sowie:  $\text{FACE}_{(16)} = ((15 \cdot 16 + 10) \cdot 16 + 12) \cdot 16 + 14 = 64206_{(10)}$ .

F	A	C	E
+	+	+	+
	240	4000	64192
=	=	=	=
15	250	4012	64206

Diagram illustrating the conversion of the hexadecimal number FACE<sub>(16)</sub> to decimal. The digits F, A, C, and E are shown above their respective place values. The calculation is shown as a sequence of additions and multiplications by 16, indicated by arrows:

- $15 \cdot 16 = 240$
- $240 + 10 = 250$
- $250 \cdot 16 = 4000$
- $4000 + 12 = 4012$
- $4012 \cdot 16 = 64192$
- $64192 + 14 = 64206$



- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0$$

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

$$3 \div 2 = 1 \quad 3 \bmod 2 = 1$$



- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

$$3 \div 2 = 1 \quad 3 \bmod 2 = 1$$

$$1 \div 2 = 0 \quad 1 \bmod 2 = 1$$

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0 \quad (\leftarrow \text{LSB})$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

$$3 \div 2 = 1 \quad 3 \bmod 2 = 1$$

$$1 \div 2 = 0 \quad 1 \bmod 2 = 1 \quad (\leftarrow \text{MSB})$$

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0 \quad (\leftarrow \text{LSB})$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

$$3 \div 2 = 1 \quad 3 \bmod 2 = 1$$

$$1 \div 2 = 0 \quad 1 \bmod 2 = 1 \quad (\leftarrow \text{MSB})$$

- Damit gilt:  $1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0$

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0 \quad (\leftarrow \text{LSB})$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

$$3 \div 2 = 1 \quad 3 \bmod 2 = 1$$

$$1 \div 2 = 0 \quad 1 \bmod 2 = 1 \quad (\leftarrow \text{MSB})$$

- Damit gilt:  $1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}.$

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0 \quad (\leftarrow \text{LSB})$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

$$3 \div 2 = 1 \quad 3 \bmod 2 = 1$$

$$1 \div 2 = 0 \quad 1 \bmod 2 = 1 \quad (\leftarrow \text{MSB})$$

- Damit gilt:  $1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}$ .
- *Hinweis: Für b2 genügt der Test auf gerade oder ungerade.*

- $14_{(10)}$  zur Basis 2. Eine Vorbemerkung:

$$14 \div 2 = 7 \quad 14 \bmod 2 = 0 \quad (\leftarrow \text{LSB})$$

$$7 \div 2 = 3 \quad 7 \bmod 2 = 1$$

$$3 \div 2 = 1 \quad 3 \bmod 2 = 1$$

$$1 \div 2 = 0 \quad 1 \bmod 2 = 1 \quad (\leftarrow \text{MSB})$$

- Damit gilt:  $1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}$ .
- *Hinweis: Für b2 genügt der Test auf gerade oder ungerade.*
- Dieses Art der Darstellung ist aber *nicht* das Hornerschema!

# Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch.

# Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch. Zur Erinnerung:



# Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch. Zur Erinnerung:

$$1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}.$$

# Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch. Zur Erinnerung:

$$1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}.$$

$$\begin{array}{r} 0 \\ + \\ 14 \\ = \\ 14 \end{array}$$

# Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch. Zur Erinnerung:

$$1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}.$$

1		0
+		+
6		14
=	$\swarrow \div 2$	=
7		14

# Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch. Zur Erinnerung:

$$1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}.$$

1	1	0
+	+	+
2	6	14
=	=	=
3	7	14

Diagram illustrating the conversion of binary 1110 to decimal 14 using a table. The table shows the intermediate results of the calculation: 1, 1, 0; 1, 1, 0; 2, 6, 14; and 3, 7, 14. Arrows indicate the progression of the calculation, with labels  $\div 2$  and  $\div 2$  indicating the division by 2 step.

# Übungsblatt 1 - Aufgabe 1 b)

- Natürlich geht dies auch tabellarisch. Zur Erinnerung:

$$1110_{(2)} = (((1 \cdot 2 + 1) \cdot 2 + 1) \cdot 2 + 0 = 14_{(10)}.$$

1	1	1	0
+	+	+	+
=	2	6	14
$\swarrow \div 2$	$\swarrow \div 2$	$\swarrow \div 2$	
1	3	7	14

# Übungsblatt 1 - Aufgabe 2

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer Liste von Tagespreisen des Tablets den maximalen prozentualen Preisanstieg von zwei aufeinanderfolgenden Tagen berechnet.

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer Liste von Tagespreisen des Tablets den maximalen prozentualen Preisanstieg von zwei aufeinanderfolgenden Tagen berechnet.



# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den **maximalen prozentualen Preisanstieg** von **zwei aufeinanderfolgenden Tagen** berechnet.

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den **maximalen prozentualen Preisanstieg von zwei aufeinanderfolgenden Tagen** berechnet.

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den **maximalen prozentualen Preisanstieg von zwei aufeinanderfolgenden Tagen** berechnet.

Liste von Tagespreise

$(p_1, \dots, p_n)_{n \in \mathbb{N}}$

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den **maximalen prozentualen Preisanstieg** von **zwei aufeinanderfolgenden Tagen** berechnet.

Liste von Tagespreise

$(p_1, \dots, p_n)_{n \in \mathbb{N}}$



Max. % Preisanstieg

$\Delta_{\max}$

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

**Begriffe:**

Entwickeln Sie einen Algorithmus, der aus einer Liste von Tagespreisen des Tablets den maximalen prozentualen Preisanstieg von zwei aufeinanderfolgenden Tagen berechnet.

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen Tagespreise eines Tablets über die letzten Monate liefert. Sie möchten nun den maximalen prozentualen Preisanstieg zwischen zwei aufeinanderfolgenden Tagen für dieses Tablet herausfinden.

Begriffe:

Entwickeln Sie einen Algorithmus, der aus einer Liste von Tagespreisen des Tablets den maximalen prozentualen Preisanstieg von zwei aufeinanderfolgenden Tagen berechnet.

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen **Tagespreise** eines Tablets über die letzten Monate liefert. Sie möchten nun den **maximalen prozentualen Preisanstieg** zwischen **zwei aufeinanderfolgenden Tagen** für dieses Tablet herausfinden.

Begriffe:

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den maximalen prozentualen Preisanstieg von **zwei aufeinanderfolgenden Tagen** berechnet.

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen **Tagespreise** eines Tablets über die letzten Monate liefert. Sie möchten nun den **maximalen prozentualen Preisanstieg** zwischen **zwei aufeinanderfolgenden Tagen** für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den maximalen prozentualen Preisanstieg von **zwei aufeinanderfolgenden Tagen** berechnet.

Begriffe:  
Tagespreisliste



# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen **Tagespreise** eines Tablets über die letzten Monate liefert. Sie möchten nun den **maximalen prozentualen Preisanstieg** zwischen **zwei aufeinanderfolgenden Tagen** für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den maximalen prozentualen Preisanstieg von **zwei aufeinanderfolgenden Tagen** berechnet.

Begriffe:

Tagespreisliste  
zwei auff. Tage

# Übungsblatt 1 - Aufgabe 2

Angenommen, Sie überlegen sich ein neues Tablet für die Uni zu kaufen. Sie stellen jedoch fest, dass die Preise für elektronische Geräte durch den Chipmangel in letzter Zeit stark gestiegen sind. Nun haben Sie eine Website gefunden, die Ihnen die jeweiligen **Tagespreise** eines Tablets über die letzten Monate liefert. Sie möchten nun den **maximalen prozentualen Preisanstieg** zwischen **zwei aufeinanderfolgenden Tagen** für dieses Tablet herausfinden.

Entwickeln Sie einen Algorithmus, der aus einer **Liste von Tagespreisen** des Tablets den maximalen prozentualen Preisanstieg von **zwei aufeinanderfolgenden Tagen** berechnet.

Begriffe:

Tagespreisliste  
zwei auff. Tage  
max. % Preisanstieg

# Übungsblatt 1 - Aufgabe 2 a) & b)

# Übungsblatt 1 - Aufgabe 2 a) & b)

a) Problemspezifikation:

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage*: zwei aufeinanderfolgende Listenindizes:  $i$  und  $i + 1$  wobei  $i, i + 1 < n$ .



# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage*: zwei aufeinanderfolgende Listenindizes:  $i$  und  $i + 1$  wobei  $i, i + 1 < n$ .
- *Preisanstieg*: positive Veränderung einer reellen Zahl (dem Tagespreis):  $p_{i+1} - p_i > 0$ .

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage*: zwei aufeinanderfolgende Listenindizes:  $i$  und  $i + 1$  wobei  $i, i + 1 < n$ .
- *Preisanstieg*: positive Veränderung einer reellen Zahl (dem Tagespreis):  $p_{i+1} - p_i > 0$ .
- *maximaler prozentualer Preisanstieg*: größter relativer Preisanstieg:  $\max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i} \cdot 100$ .

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage*: zwei aufeinanderfolgende Listenindizes:  $i$  und  $i + 1$  wobei  $i, i + 1 < n$ .
- *Preisanstieg*: positive Veränderung einer reellen Zahl (dem Tagespreis):  $p_{i+1} - p_i > 0$ .
- *maximaler prozentualer Preisanstieg*: größter relativer Preisanstieg:  $\max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i} \cdot 100$ .

**Wichtig:** Wir sind hier nicht in einer Programmiersprache: Typen wie „int“ gibt es nicht  $\Rightarrow$  mathematische Notation benutzen oder eigenen Typ definieren (wie Tupel, ...)!

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage*: zwei aufeinanderfolgende Listenindizes:  $i$  und  $i + 1$  wobei  $i, i + 1 < n$ .
- *Preisanstieg*: positive Veränderung einer reellen Zahl (dem Tagespreis):  $p_{i+1} - p_i > 0$ .
- *maximaler prozentualer Preisanstieg*: größter relativer Preisanstieg:  $\max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i} \cdot 100$ .

**Wichtig:** Wir sind hier nicht in einer Programmiersprache: Typen wie „int“ gibt es nicht  $\Rightarrow$  mathematische Notation benutzen oder eigenen Typ definieren (wie Tupel, ...)!

## b) Problemabstraktion:

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage*: zwei aufeinanderfolgende Listenindizes:  $i$  und  $i + 1$  wobei  $i, i + 1 < n$ .
- *Preisanstieg*: positive Veränderung einer reellen Zahl (dem Tagespreis):  $p_{i+1} - p_i > 0$ .
- *maximaler prozentualer Preisanstieg*: größter relativer Preisanstieg:  $\max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i} \cdot 100$ .

**Wichtig:** Wir sind hier nicht in einer Programmiersprache: Typen wie „int“ gibt es nicht  $\Rightarrow$  mathematische Notation benutzen oder eigenen Typ definieren (wie Tupel, ...)!

## b) Problemabstraktion:

- *Gegeben*: Chronologisch sortierte Liste  $[p_1, \dots, p_n]$  positiver reeller Zahlen. Annahme:  $n > 1$  („letzte Monate“).

# Übungsblatt 1 - Aufgabe 2 a) & b)

## a) Problemspezifikation:

- *Tagespreis*: positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste*: chronologisch sortierte Liste an Tagespreisen  $[p_1, \dots, p_n]$  mit  $n \in \mathbb{N}_1$ .
- *Tag*: natürlicher Index in Tagespreisliste.
- *Zwei aufeinanderfolgende Tage*: zwei aufeinanderfolgende Listenindizes:  $i$  und  $i + 1$  wobei  $i, i + 1 < n$ .
- *Preisanstieg*: positive Veränderung einer reellen Zahl (dem Tagespreis):  $p_{i+1} - p_i > 0$ .
- *maximaler prozentualer Preisanstieg*: größter relativer Preisanstieg:  $\max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i} \cdot 100$ .

**Wichtig:** Wir sind hier nicht in einer Programmiersprache: Typen wie „int“ gibt es nicht  $\Rightarrow$  mathematische Notation benutzen oder eigenen Typ definieren (wie Tupel, ...)!

## b) Problemabstraktion:

- *Gegeben*: Chronologisch sortierte Liste  $[p_1, \dots, p_n]$  positiver reeller Zahlen. Annahme:  $n > 1$  („letzte Monate“).
- *Gesucht*: Positive reelle Zahl  $m = 100 \cdot \max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i}$  ( $X \% \hat{=} \frac{X}{100}$ ).

# Übungsblatt 1 - Aufgabe 2 c) & d)

c) Algorithmenentwurf:

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

(1) Setze  $\Delta_{\max} = 0$ .



# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## d) Korrektheitsnachweis, Verifikation:

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## d) Korrektheitsnachweis, Verifikation:

1. Formale vollständige Induktion, oder:



# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## d) Korrektheitsnachweis, Verifikation:

1. Formale vollständige Induktion, oder:
2. „Textbasiert“.

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## d) Korrektheitsnachweis, Verifikation:

1. Formale vollständige Induktion, oder:
2. „Textbasiert“.  $i$  wächst streng monoton an, die Schleife wird damit genau  $n - 1$  mal durchlaufen.

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## d) Korrektheitsnachweis, Verifikation:

1. Formale vollständige Induktion, oder:
2. „Textbasiert“.  $i$  wächst streng monoton an, die Schleife wird damit genau  $n - 1$  mal durchlaufen. Alle mathematischen Berechnungen terminieren per Konstruktion, ebenso die Zuweisungen. Der Algorithmus *terminiert*.

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## d) Korrektheitsnachweis, Verifikation:

1. Formale vollständige Induktion, oder:
2. „Textbasiert“.  $i$  wächst streng monoton an, die Schleife wird damit genau  $n - 1$  mal durchlaufen. Alle mathematischen Berechnungen terminieren per Konstruktion, ebenso die Zuweisungen. Der Algorithmus *terminiert*.

Zudem ist er *partiell korrekt*. Die maximale prozentuale Änderung ist immer größte relative Differenz für alle  $[p_1, \dots, p_{i+1}]$  im  $i$ -ten Durchlauf.

# Übungsblatt 1 - Aufgabe 2 c) & d)

## c) Algorithmenentwurf:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## d) Korrektheitsnachweis, Verifikation:

1. Formale vollständige Induktion, oder:
2. „Textbasiert“.  $i$  wächst streng monoton an, die Schleife wird damit genau  $n - 1$  mal durchlaufen. Alle mathematischen Berechnungen terminieren per Konstruktion, ebenso die Zuweisungen. Der Algorithmus *terminiert*.

Zudem ist er *partiell korrekt*. Die maximale prozentuale Änderung ist immer größte relative Differenz für alle  $[p_1, \dots, p_{i+1}]$  im  $i$ -ten Durchlauf. Nach  $n - 1$  Durchläufen:  $[p_1, \dots, p_n]$ . Basisfall mit  $i = 2$ , für Schritt  $i \rightarrow i + 1$ .



- Totale Korrektheit erfordert zwei Komponenten!

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.



- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- Es reicht in der Regel nicht aus:

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“.

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“.

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.
- **Vollständige Induktion über  $i$  (meist bei Schleifen):**

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.
- **Vollständige Induktion über  $i$  (meist bei Schleifen):**
  - Achtet darauf klar anzugeben, über was die Induktion läuft



- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.
- **Vollständige Induktion über  $i$  (meist bei Schleifen):**
  - Achtet darauf klar anzugeben, über was die Induktion läuft (Länge der Einladungs-, Bekannten- oder Bedingungsliste? Das Durchschnittsalter der Person?)

- **Totale Korrektheit erfordert zwei Komponenten!**

1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.

- **Es reicht in der Regel nicht aus:**

- „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
- „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.

- **Vollständige Induktion über  $i$  (meist bei Schleifen):**

- Achtet darauf klar anzugeben, über was die Induktion läuft (Länge der Einladungs-, Bekannten- oder Bedingungsliste? Das Durchschnittsalter der Person?)
- Achtet auf eine vollständige Behauptung. Ist die unvollständig, ist es auch der Prozess an sich.

### ■ Totale Korrektheit erfordert zwei Komponenten!

1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.

### ■ Es reicht in der Regel nicht aus:

- „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
- „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.

### ■ Vollständige Induktion über $i$ (meist bei Schleifen):

- Achtet darauf klar anzugeben, über was die Induktion läuft (Länge der Einladungs-, Bekannten- oder Bedingungsliste? Das Durchschnittsalter der Person?)
- Achtet auf eine vollständige Behauptung. Ist die unvollständig, ist es auch der Prozess an sich.
- Induktionsanfang: Wir zeigen, dass es für ein  $i \in \mathbb{N}$  gilt, in der Regel  $i = 0$  oder  $i = 1$ .

### ■ Totale Korrektheit erfordert zwei Komponenten!

1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.

### ■ Es reicht in der Regel nicht aus:

- „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
- „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.

### ■ Vollständige Induktion über $i$ (meist bei Schleifen):

- Achtet darauf klar anzugeben, über was die Induktion läuft (Länge der Einladungs-, Bekannten- oder Bedingungsliste? Das Durchschnittsalter der Person?)
- Achtet auf eine vollständige Behauptung. Ist die unvollständig, ist es auch der Prozess an sich.
- Induktionsanfang: Wir zeigen, dass es für ein  $i \in \mathbb{N}$  gilt, in der Regel  $i = 0$  oder  $i = 1$ .
- Induktionshypothese: Die Behauptung gilt für (ein)  $n$ .

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.
- **Vollständige Induktion über  $i$  (meist bei Schleifen):**
  - Achtet darauf klar anzugeben, über was die Induktion läuft (Länge der Einladungs-, Bekannten- oder Bedingungsliste? Das Durchschnittsalter der Person?)
  - Achtet auf eine vollständige Behauptung. Ist die unvollständig, ist es auch der Prozess an sich.
  - Induktionsanfang: Wir zeigen, dass es für ein  $i \in \mathbb{N}$  gilt, in der Regel  $i = 0$  oder  $i = 1$ .
  - Induktionshypothese: Die Behauptung gilt für (ein)  $n$ .
  - Induktionsschritt: Gilt die Behauptung für  $n$ , so gilt sie auch für  $n + 1$ .

- **Totale Korrektheit erfordert zwei Komponenten!**
  1. *Terminiertheit*: Der Algorithmus terminiert für jede definierte Eingabe.
  2. *Partielle Korrektheit*: Der Algorithmus liefert für jede definierte Eingabe ein korrektes Ergebnis, sofern er terminiert.
- **Es reicht in der Regel nicht aus:**
  - „Maximum ist sicher größer als alle betrachteten Alternativen“. Es muss dann zum Beispiel auch gezeigt werden, dass alle relevanten Alternativen in Betracht gezogen werden.
  - „Der Algorithmus findet das gesuchte Ergebnis“. Wir Beweisen zwar (noch) nicht ganz formal. Dennoch sollte ein ausreichendes Verständnis für den Beweis gezeigt werden.
- **Vollständige Induktion über  $i$  (meist bei Schleifen):**
  - Achtet darauf klar anzugeben, über was die Induktion läuft (Länge der Einladungs-, Bekannten- oder Bedingungsliste? Das Durchschnittsalter der Person?)
  - Achtet auf eine vollständige Behauptung. Ist die unvollständig, ist es auch der Prozess an sich.
  - Induktionsanfang: Wir zeigen, dass es für ein  $i \in \mathbb{N}$  gilt, in der Regel  $i = 0$  oder  $i = 1$ .
  - Induktionshypothese: Die Behauptung gilt für (ein)  $n$ .
  - Induktionsschritt: Gilt die Behauptung für  $n$ , so gilt sie auch für  $n + 1$ . (Hier wird fast immer die Hypothese benutzt um den Fall  $n + 1$  auf  $n$  zurückzuführen.)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.



- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange  $(i < n)$ , wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ ,

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange  $(i < n)$ , wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

IH: Es gilt  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  für  $i$ .

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

IH: Es gilt  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  für  $i$ .

IS: Wir zeigen, dass auch  $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$  gilt.

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

IH: Es gilt  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  für  $i$ .

IS: Wir zeigen, dass auch  $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$  gilt. Das heißt, aktuell ist  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  und in der weiteren Schleife gilt:

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

IH: Es gilt  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  für  $i$ .

IS: Wir zeigen, dass auch  $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$  gilt. Das heißt, aktuell ist  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  und in der weiteren Schleife gilt:

1.  $\text{test} > \Delta_{\max}$ : Durch die Bedingung wird  $\Delta_{\max}$  aktualisiert. Es gilt:  $\Delta_{\max} = \text{test} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$ .



- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:
  - Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .
  - Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .
  - Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

IH: Es gilt  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  für  $i$ .

IS: Wir zeigen, dass auch  $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$  gilt. Das heißt, aktuell ist  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  und in der weiteren Schleife gilt:

1.  $\text{test} > \Delta_{\max}$ : Durch die Bedingung wird  $\Delta_{\max}$  aktualisiert. Es gilt:  $\Delta_{\max} = \text{test} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$ .
2.  $\text{test} \leq \Delta_{\max}$ : Das neue Glied durch  $i + 1$  ist kein neues Maximum. Es gilt:  
 $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j) \geq \text{test}$ .

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:
  - Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .
  - Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .
  - Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

IH: Es gilt  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  für  $i$ .

IS: Wir zeigen, dass auch  $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$  gilt. Das heißt, aktuell ist  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  und in der weiteren Schleife gilt:

1.  $\text{test} > \Delta_{\max}$ : Durch die Bedingung wird  $\Delta_{\max}$  aktualisiert. Es gilt:  $\Delta_{\max} = \text{test} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$ .
2.  $\text{test} \leq \Delta_{\max}$ : Das neue Glied durch  $i + 1$  ist kein neues Maximum. Es gilt:  
 $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j) \geq \text{test}$ .

Nach  $n - 1$  Schritten durch die Begrenzung  $i < n$ , wurden alle  $p_1, \dots, p_n$  betrachtet.

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:
  - Setze  $\text{test} = (p_{i+1} - p_i)/p_i$ .
  - Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .
  - Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

Wir zeigen die partielle Korrektheit und nehmen die Termination hier als gegeben.

Für ein beliebiges aber festes  $n \in \mathbb{N}$  mit  $n \geq 2$  und der Eingabe  $p_1, \dots, p_n$  zeigen wir per vollständiger Induktion über  $i$ , dass in jedem Schritte gelte:  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$ .

IA: Mit  $i = 1$  ist  $\Delta_{\max} = \frac{p_2 - p_1}{p_1}$  das triviale Maximum ( $1 \leq j \leq 1$ ).

IH: Es gilt  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  für  $i$ .

IS: Wir zeigen, dass auch  $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$  gilt. Das heißt, aktuell ist  $\Delta_{\max} \geq \max_{1 \leq j \leq i} ((p_{j+1} - p_j)/p_j)$  und in der weiteren Schleife gilt:

1.  $\text{test} > \Delta_{\max}$ : Durch die Bedingung wird  $\Delta_{\max}$  aktualisiert. Es gilt:  $\Delta_{\max} = \text{test} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j)$ .
2.  $\text{test} \leq \Delta_{\max}$ : Das neue Glied durch  $i + 1$  ist kein neues Maximum. Es gilt:  
 $\Delta_{\max} \geq \max_{1 \leq j \leq i+1} ((p_{j+1} - p_j)/p_j) \geq \text{test}$ .

Nach  $n - 1$  Schritten durch die Begrenzung  $i < n$ , wurden alle  $p_1, \dots, p_n$  betrachtet.  $\Delta_{\max}$  ist Maximum aller.

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

e) Aufwandsanalyse:

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.



# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
  - Die äußere Schleife in 3 wird genau  $n - 1$  mal durchlaufen.

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
  - Die äußere Schleife in 3 wird genau  $n - 1$  mal durchlaufen.
    - Erste Schleifenanweisung bedarf dreier Elementaroperationen: Subtraktion, Division & Zuweisung.

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
  - Die äußere Schleife in 3 wird genau  $n - 1$  mal durchlaufen.
    - Erste Schleifenanweisung bedarf dreier Elementaroperationen: Subtraktion, Division & Zuweisung.
    - Zweite Schleifenanweisung ist ein Vergleich und maximal eine Zuweisung.

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
  - Die äußere Schleife in 3 wird genau  $n - 1$  mal durchlaufen.
    - Erste Schleifenanweisung bedarf dreier Elementaroperationen: Subtraktion, Division & Zuweisung.
    - Zweite Schleifenanweisung ist ein Vergleich und maximal eine Zuweisung.
    - Dritte Schleifenanweisung ist eine Elementaroperation (je nach Definition auch zwei).

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
  - Die äußere Schleife in 3 wird genau  $n - 1$  mal durchlaufen.
    - Erste Schleifenanweisung bedarf dreier Elementaroperationen: Subtraktion, Division & Zuweisung.
    - Zweite Schleifenanweisung ist ein Vergleich und maximal eine Zuweisung.
    - Dritte Schleifenanweisung ist eine Elementaroperation (je nach Definition auch zwei).
  - Das Endergebnis ist eine Multiplikation (und je nach Definition eine Zuweisung).

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
  - Die äußere Schleife in 3 wird genau  $n - 1$  mal durchlaufen.
    - Erste Schleifenanweisung bedarf dreier Elementaroperationen: Subtraktion, Division & Zuweisung.
    - Zweite Schleifenanweisung ist ein Vergleich und maximal eine Zuweisung.
    - Dritte Schleifenanweisung ist eine Elementaroperation (je nach Definition auch zwei).
  - Das Endergebnis ist eine Multiplikation (und je nach Definition eine Zuweisung).
3. Damit erhalten wir:  $2 + (n - 1) \cdot (3 + 2 + 1) + 2 + 2 = 6n - 2$

# Übungsblatt 1 - Aufgabe 2 e)

- (1) Setze  $\Delta_{\max} = 0$ .
- (2) Setze  $i = 1$ .
- (3) Solange ( $i < n$ ), wiederhole:  
    Setze  $\text{test} = (p_{i+1} - p_i) / p_i$ .  
    Wenn  $\text{test} > \Delta_{\max}$ : Setze  $\Delta_{\max} = \text{test}$ .  
    Erhöhe  $i$  um 1.
- (4) Ergebnis ist  $100 \cdot \Delta_{\max}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die ersten Zuweisungen 1 & 2 sind exakt zwei Elementaroperation.
  - Die äußere Schleife in 3 wird genau  $n - 1$  mal durchlaufen.
    - Erste Schleifenanweisung bedarf dreier Elementaroperationen: Subtraktion, Division & Zuweisung.
    - Zweite Schleifenanweisung ist ein Vergleich und maximal eine Zuweisung.
    - Dritte Schleifenanweisung ist eine Elementaroperation (je nach Definition auch zwei).
  - Das Endergebnis ist eine Multiplikation (und je nach Definition eine Zuweisung).
3. Damit erhalten wir:  $2 + (n - 1) \cdot (3 + 2 + 1) + 2 + 2 = 6n - 2$  (also  $O(n)$ ).

# Aussicht: Das IEEE 754 Format



# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$
- Grundlegend gilt für eine Zahl  $a = s \cdot m \cdot b^e$ , wobei:

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$
- Grundslegend gilt für eine Zahl  $a = s \cdot m \cdot b^e$ , wobei:
  - s Signum, Vorzeichen, Plus oder Minus

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$
- Grundslegend gilt für eine Zahl  $a = s \cdot m \cdot b^e$ , wobei:
  - s Signum, Vorzeichen, Plus oder Minus
  - m Mantisse an | m | Stellen (wie 1.42), später genauer

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$
- Grundslegend gilt für eine Zahl  $a = s \cdot m \cdot b^e$ , wobei:
  - s Signum, Vorzeichen, Plus oder Minus
  - m Mantisse an  $|m|$  Stellen (wie 1.42), später genauer
  - b Basis, normiert, für uns: 10, im IEEE 754: 2

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$
- Grundslegend gilt für eine Zahl  $a = s \cdot m \cdot b^e$ , wobei:
  - s Signum, Vorzeichen, Plus oder Minus
  - m Mantisse an  $|m|$  Stellen (wie 1.42), später genauer
  - b Basis, normiert, für uns: 10, im IEEE 754: 2
  - e Exponent an  $|e|$  Stellen.



# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$
- Grundslegend gilt für eine Zahl  $a = s \cdot m \cdot b^e$ , wobei:
  - s Signum, Vorzeichen, Plus oder Minus
  - m Mantisse an  $|m|$  Stellen (wie 1.42), später genauer
  - b Basis, normiert, für uns: 10, im IEEE 754: 2
  - e Exponent an  $|e|$  Stellen.
- Da Exponent immer positiv, existiert noch ein Bias B, ebenfalls definiert (meist auf die „Hälfte“ des maximal möglichen Exponenten)

# Aussicht: Das IEEE 754 Format

- Erlaubt es Gleitkommazahl in Bits darzustellen
- Vorteil zu Festkommaformat: flexibler
- Beispiel mit wissenschaftlicher Notation:  $+1.42 \cdot 10^{12}$
- Grundslegend gilt für eine Zahl  $a = s \cdot m \cdot b^e$ , wobei:
  - s Signum, Vorzeichen, Plus oder Minus
  - m Mantisse an  $|m|$  Stellen (wie 1.42), später genauer
  - b Basis, normiert, für uns: 10, im IEEE 754: 2
  - e Exponent an  $|e|$  Stellen.
- Da Exponent immer positiv, existiert noch ein Bias B, ebenfalls definiert (meist auf die „Hälfte“ das maximal möglichen Exponenten)
- Die Mantisse wird für  $b = 2$  auf 1. („Eins komma“) normiert, es werden nur die Stellen nach dem Komma gespeichert.

# Das IEEE 754 Format - Ein Beispiel

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
  3. Normalisiere Mantisse:  $111.01 =$



# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
  3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
  3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
  4. Berechne Exponent:  $e = 2 \xrightarrow{+B=1}$

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
  3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
  4. Berechne Exponent:  $e = 2 \xrightarrow{+B=1} E = 3_{(10)} = 11_{(2)}$

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
  3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
  4. Berechne Exponent:  $e = 2^{+B=1} \Rightarrow E = 3_{(10)} = 11_{(2)}$
  5. Überraschung, es geht alles auf ☺ (wenn nicht, muss dies kenntlich gemacht werden, es existieren Rundungsverfahren!)

# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:
  1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
  2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
  3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
  4. Berechne Exponent:  $e = 2^{+B=1} \Rightarrow E = 3_{(10)} = 11_{(2)}$
  5. Überraschung, es geht alles auf ☺ (wenn nicht, muss dies kenntlich gemacht werden, es existieren Rundungsverfahren!)
  6. Gebe Ergebnis an ( $S = 0$ ,  $E = 11$ ,  $M = 1101$ ).

# Das IEEE 754 Format - Ein Beispiel

## ■ Konvertiere 7.25 ins IEEE 754 Format, wobei $|m| = 5$ , $|e| = 2$ . Der Bias sei 1:

1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
4. Berechne Exponent:  $e = 2^{+B=1} \Rightarrow E = 3_{(10)} = 11_{(2)}$
5. Überraschung, es geht alles auf ☺ (wenn nicht, muss dies kenntlich gemacht werden, es existieren Rundungsverfahren!)
6. Gebe Ergebnis an ( $S = 0$ ,  $E = 11$ ,  $M = 1101$ ). Wir erhalten:

$$\underbrace{0}_S \underbrace{11}_E \underbrace{1101}_M = 01111010$$

# Das IEEE 754 Format - Ein Beispiel

## ■ Konvertiere 7.25 ins IEEE 754 Format, wobei $|m| = 5$ , $|e| = 2$ . Der Bias sei 1:

1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
4. Berechne Exponent:  $e = 2^{+B=1} \Rightarrow E = 3_{(10)} = 11_{(2)}$
5. Überraschung, es geht alles auf ☺ (wenn nicht, muss dies kenntlich gemacht werden, es existieren Rundungsverfahren!)
6. Gebe Ergebnis an ( $S = 0$ ,  $E = 11$ ,  $M = 1101$ ). Wir erhalten:

$$\underbrace{0}_S \underbrace{11}_E \underbrace{11010}_M = 01111010$$

Die Mantisse wird logisch (rechts!!!) mit Nullen aufgefüllt (8 Bit, jay ☺).

# Das IEEE 754 Format - Ein Beispiel

## ■ Konvertiere 7.25 ins IEEE 754 Format, wobei $|m| = 5$ , $|e| = 2$ . Der Bias sei 1:

1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
4. Berechne Exponent:  $e = 2^{+B=1} \Rightarrow E = 3_{(10)} = 11_{(2)}$
5. Überraschung, es geht alles auf ☺ (wenn nicht, muss dies kenntlich gemacht werden, es existieren Rundungsverfahren!)
6. Gebe Ergebnis an ( $S = 0$ ,  $E = 11$ ,  $M = 1101$ ). Wir erhalten:

$$\underbrace{0}_S \underbrace{11}_E \underbrace{1101}_M = 01111010$$

Die Mantisse wird logisch (rechts!!!) mit Nullen aufgefüllt (8 Bit, jay ☺).

7. Hinweis: Zahlen wie die '0', Unendlich, NaN (Not a Number) besitzen besondere Darstellungen.



# Das IEEE 754 Format - Ein Beispiel

- Konvertiere 7.25 ins IEEE 754 Format, wobei  $|m| = 5$ ,  $|e| = 2$ . Der Bias sei 1:

1. positiv  $\Rightarrow S = 0$  [ $s = (-1)^S$ ]
2. konvertiere die Zahl in das Binärsystem (nach dem Komma gilt  $2^{-1}2^{-2}2^{-3} \dots$ ):  $7.25_{(10)} = 111.01$
3. Normalisiere Mantisse:  $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
4. Berechne Exponent:  $e = 2^{+B=1} \Rightarrow E = 3_{(10)} = 11_{(2)}$
5. Überraschung, es geht alles auf ☺ (wenn nicht, muss dies kenntlich gemacht werden, es existieren Rundungsverfahren!)
6. Gebe Ergebnis an ( $S = 0$ ,  $E = 11$ ,  $M = 1101$ ). Wir erhalten:

$$\underbrace{0}_S \underbrace{11}_E \underbrace{1101}_M = 01111010$$

Die Mantisse wird logisch (rechts!!!) mit Nullen aufgefüllt (8 Bit, jay ☺).

7. Hinweis: Zahlen wie die '0', Unendlich, NaN (Not a Number) besitzen besondere Darstellungen.
- Beachte *Verlust*, Wenn Mantisse oder Exponent zu groß/klein!

# Das IEEE 754 Format - Die Formate

# Das IEEE 754 Format - Die Formate

Hier die wichtigsten IEEE 754 Formate (dies so wirklich gibt) zum Sehen und Vergessen:

# Das IEEE 754 Format - Die Formate

Hier die wichtigsten IEEE 754 Formate (dies so wirklich gibt) zum Sehen und Vergessen:

Typ	Größe	Exponent	Mantisse	Bias
single	32 bit	8 bit	23 bit	127
double	64 bit	11 bit	52 bit	1023
⋮				
single ext, min	43 bit	11 bit	31 bit	1023
double ext. min	79 bit	15 bit	63 bit	16383

# Das IEEE 754 Format - Ein Beispiel, reloaded

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$



# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an:

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010 multipliziere über Basis (= 2) mit Exponent:

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010 multipliziere über Basis (= 2) mit Exponent:  $1.1101 \cdot 2^2$

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010 multipliziere über Basis (= 2) mit Exponent:  $1.1101 \cdot 2^2 = 111.01$ .

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010 multipliziere über Basis (= 2) mit Exponent:  $1.1101 \cdot 2^2 = 111.01$ .
  4. Rechne um:  $111.01_{(2)}$

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010 multipliziere über Basis (= 2) mit Exponent:  $1.1101 \cdot 2^2 = 111.01$ .
  4. Rechne um:  $111.01_{(2)} \hat{=} 7 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 7.25_{(10)}$



# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010 multipliziere über Basis (= 2) mit Exponent:  $1.1101 \cdot 2^2 = 111.01$ .
  4. Rechne um:  $111.01_{(2)} \hat{=} 7 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 7.25_{(10)}$
  5. Füge Vorzeichen an (hier positiv):

# Das IEEE 754 Format - Ein Beispiel, reloaded

- Konvertiere 01111010 aus dem IEEE 754 Format ins Fließkommaformat, wobei  $|m| = 5$  und  $|e| = 2$ . Der Bias sei 1:
  1. Erstes Bit 0  $\Rightarrow$  positiv
  2. Betrachte nächste beiden Bits (da  $|e| = 2$ ):  $11_{(2)} \hat{=} 3_{(10)}$  Bias abziehen ergibt:  $3 - 1 = 2$ .
  3. Isoliere Mantisse: 11010 (da  $|m| = 5$ ), füge „Eins komma“ an: 1.11010 multipliziere über Basis (= 2) mit Exponent:  $1.1101 \cdot 2^2 = 111.01$ .
  4. Rechne um:  $111.01_{(2)} \hat{=} 7 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 7.25_{(10)}$
  5. Füge Vorzeichen an (hier positiv): 7.25, ferddisch! (und schdimmd, 😊)

# Das IEEE 754 Format - Rundungsproblematik

# Das IEEE 754 Format - Rundungsproblematik

- Da mit  $2^{-1}2^{-2}2^{-3} \dots$  nicht jede (dezimale) Fließkommazahl exakt darstellbar.

# Das IEEE 754 Format - Rundungsproblematik

- Da mit  $2^{-1}2^{-2}2^{-3} \dots$  nicht jede (dezimale) Fließkommazahl exakt darstellbar.
- Beispiel:

# Das IEEE 754 Format - Rundungsproblematik

- Da mit  $2^{-1}2^{-2}2^{-3} \dots$  nicht jede (dezimale) Fließkommazahl exakt darstellbar.
- Beispiel:

```
1 class RoundingProblem {  
2     public static void main(String[] args) {  
3         double money = 0.1D;  
4         System.out.println(money);  
5         System.out.println(1.0 + money - 1.0);  
6     }  
7 }
```

# Das IEEE 754 Format - Rundungsproblematik

- Da mit  $2^{-1}2^{-2}2^{-3} \dots$  nicht jede (dezimale) Fließkommazahl exakt darstellbar.
- Beispiel:

```
1 class RoundingProblem {  
2     public static void main(String[] args) {  
3         double money = 0.1D;  
4         System.out.println(money);  
5         System.out.println(1.0 + money - 1.0);  
6     }  
7 }
```

- Erwartung: 0.1 und 0.1

# Das IEEE 754 Format - Rundungsproblematik

- **Leider nein:** `java` RoundingProblem:



# Das IEEE 754 Format - Rundungsproblematik

- **Leider nein:** `java` RoundingProblem:

*0.1*

*0.100000000000000009*

# Das IEEE 754 Format - Rundungsproblematik

- **Leider nein:** `java` RoundingProblem:

*0.1*

*0.1000000000000000009*

- Warum?

# Das IEEE 754 Format - Rundungsproblematik

- **Leider nein:** `java` RoundingProblem:

*0.1*

*0.1000000000000000009*

- **Warum?**

$0.1_{(10)}$

# Das IEEE 754 Format - Rundungsproblematik

- **Leider nein:** `java` RoundingProblem:

`0.1`

`0.100000000000000009`

- Warum?

$$0.1_{(10)} \hat{=} 0.000110011001100110011 \dots_{(2)}$$

# Das IEEE 754 Format - Rundungsproblematik

- **Leider nein:** `java` RoundingProblem:

```
0.1  
0.100000000000000009
```

- Warum?

$0.1_{(10)} \hat{=} 0.000110011001100110011 \dots_{(2)} \neq 0.1_{(10)}$  da Mantisse begrenzt.

# Algorithmenbau

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der Liste Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die möglichst viele Personen einlädt.



Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der Liste Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die möglichst viele Personen einlädt.

- a) Problemspezifikation
- b) Problemabstraktion
- c) Algorithmenentwurf
- d) Korrektheitsnachweis
- e) Aufwandsanalyse

# Algorithmenbau

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der Liste Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die möglichst viele Personen einlädt.

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste Ihrer Bekannten** und der **Liste an Bedingungen** eine **Einladungsliste berechnet**, die möglichst viele Personen einlädt.

# Algorithmenbau

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste Ihrer Bekannten** und der **Liste an Bedingungen** eine **Einladungsliste berechnet**, die möglichst viele Personen einlädt.

# Algorithmenbau

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Bekanntenliste  
( $b_1, \dots, b_n$ )

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste Ihrer Bekannten** und der **Liste an Bedingungen** eine **Einladungsliste berechnet**, die möglichst viele Personen einlädt.

# Algorithmenbau

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste Ihrer Bekannten** und der **Liste an Bedingungen** eine **Einladungsliste berechnet**, die möglichst viele Personen einlädt.

Bekanntenliste

$(b_1, \dots, b_n)$

Bedingungsliste

$(c_1, \dots, c_k)$



# Algorithmenbau

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste Ihrer Bekannten** und der **Liste an Bedingungen** eine **Einladungsliste berechnet**, die möglichst viele Personen einlädt.

Bekanntenliste

$(b_1, \dots, b_n)$

Bedingungsliste

$(c_1, \dots, c_k)$



Einladungsliste

$(e_1, \dots, e_j)$

# Algorithmenbau

Gegeben sei folgende Situation: Es soll eine kleine Party veranstaltet werden, zu der Sie möglichst viele Ihrer Bekannten einladen wollen. Jedoch gibt es hier verschiedene Bedingungen, die zu beachten sind: zum einen kommen Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte Personen nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine Einladung erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von Aussagen „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

**Begriffe:**

Entwickeln Sie einen Algorithmus, der Ihnen aus der Liste Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die möglichst viele Personen einlädt.

# Algorithmenbau

Gegeben sei folgende Situation: Es soll ein **kleine Party** veranstaltet werden, zu der Sie möglichst viele Ihrer **Bekannten** einladen wollen. Jedoch gibt es hier verschiedene **Bedingungen**, die zu beachten sind: zum einen **kommen** Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte **Personen** nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine **Einladung** erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von **Aussagen** „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Begriffe:

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste** Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die **möglichst viele** Personen einlädt.

# Algorithmenbau

Gegeben sei folgende Situation: Es soll ein **kleine Party** veranstaltet werden, zu der Sie möglichst viele Ihrer **Bekannten** einladen wollen. Jedoch gibt es hier verschiedene **Bedingungen**, die zu beachten sind: zum einen **kommen** Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte **Personen** nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine **Einladung** erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von **Aussagen** „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Begriffe:

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste** Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die **möglichst viele** Personen einlädt.

# Algorithmenbau

Gegeben sei folgende Situation: Es soll ein **kleine Party** veranstaltet werden, zu der Sie möglichst viele Ihrer **Bekannten** einladen wollen. Jedoch gibt es hier verschiedene **Bedingungen**, die zu beachten sind: zum einen **kommen** Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte **Personen** nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine **Einladung** erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von **Aussagen** „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste** Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die **möglichst viele** Personen einlädt.

Begriffe:

Bekanntenliste

# Algorithmenbau

Gegeben sei folgende Situation: Es soll ein **kleine Party** veranstaltet werden, zu der Sie möglichst viele Ihrer **Bekannten** einladen wollen. Jedoch gibt es hier verschiedene **Bedingungen**, die zu beachten sind: zum einen **kommen** Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte **Personen** nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine **Einladung** erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von **Aussagen** „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste** Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die **möglichst viele** Personen einlädt.

Begriffe:

Bekanntenliste  
Bedingungsliste

# Algorithmenbau

Gegeben sei folgende Situation: Es soll ein **kleine Party** veranstaltet werden, zu der Sie möglichst viele Ihrer **Bekannten** einladen wollen. Jedoch gibt es hier verschiedene **Bedingungen**, die zu beachten sind: zum einen **kommen** Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte **Personen** nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine **Einladung** erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von **Aussagen** „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste** Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die **möglichst viele** Personen einlädt.

Begriffe:

Bekanntenliste  
Bedingungsliste  
Einladungsliste

# Algorithmenbau

Gegeben sei folgende Situation: Es soll ein **kleine Party** veranstaltet werden, zu der Sie möglichst viele Ihrer **Bekannten** einladen wollen. Jedoch gibt es hier verschiedene **Bedingungen**, die zu beachten sind: zum einen **kommen** Einige nur dann, wenn auch eine andere Person eingeladen wird. Andere wiederum können bestimmte **Personen** nicht leiden und kommen nicht, wenn eine bestimmte Person auch eine **Einladung** erhält. Nehmen Sie an, dass sie bereits eine Liste dieser Bedingungen in Form von **Aussagen** „X kommt nur, wenn Y kommt“ oder „X kommt nicht, wenn Y kommt“ aufgeschrieben haben.

Entwickeln Sie einen Algorithmus, der Ihnen aus der **Liste** Ihrer Bekannten und der Liste an Bedingungen eine Einladungsliste berechnet, die **möglichst viele** Personen einlädt.

Begriffe:

Bekanntenliste  
Bedingungsliste  
Einladungsliste  
möglichst viele



# Algorithmenbau a) & b)

# Algorithmenbau a) & b)

a) Problemspezifikation:

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladenen Bekannter.

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladenen Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladenen Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).
- *Bedingungsliste*: eine Menge an Bedingungen.

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladener Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).
- *Bedingungsliste*: eine Menge an Bedingungen.

(Warum Mengen und nicht mehr Listen? Für jede Liste können wir Argumentieren, dass Dopplungen unsinnig sind: Doppelte Einladungen sind ebenso sinnfrei wie doppelte Bedingungen. Damit lockern wir so das Problem ein wenig auf und vereinfachen uns die Lösung.)



# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladener Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).
- *Bedingungsliste*: eine Menge an Bedingungen.

(Warum Mengen und nicht mehr Listen? Für jede Liste können wir Argumentieren, dass Dopplungen unsinnig sind: Doppelte Einladungen sind ebenso sinnfrei wie doppelte Bedingungen. Damit lockern wir so das Problem ein wenig auf und vereinfachen uns die Lösung.)

## b) Problemabstraktion:

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladener Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).
- *Bedingungsliste*: eine Menge an Bedingungen.

(Warum Mengen und nicht mehr Listen? Für jede Liste können wir Argumentieren, dass Dopplungen unsinnig sind: Doppelte Einladungen sind ebenso sinnfrei wie doppelte Bedingungen. Damit lockern wir so das Problem ein wenig auf und vereinfachen uns die Lösung.)

## b) Problemabstraktion:

- *Gegeben*:  $(n, k \in \mathbb{N})$

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladener Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).
- *Bedingungsliste*: eine Menge an Bedingungen.

(Warum Mengen und nicht mehr Listen? Für jede Liste können wir Argumentieren, dass Dopplungen unsinnig sind: Doppelte Einladungen sind ebenso sinnfrei wie doppelte Bedingungen. Damit lockern wir so das Problem ein wenig auf und vereinfachen uns die Lösung.)

## b) Problemabstraktion:

- *Gegeben*:  $(n, k \in \mathbb{N})$ 
  1. Eine endliche Menge von Bekannten  $B = \{b_1, \dots, b_n\}$  (hoffentlich mit  $B \neq \emptyset$  😊).

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladener Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).
- *Bedingungsliste*: eine Menge an Bedingungen.

(Warum Mengen und nicht mehr Listen? Für jede Liste können wir Argumentieren, dass Dopplungen unsinnig sind: Doppelte Einladungen sind ebenso sinnfrei wie doppelte Bedingungen. Damit lockern wir so das Problem ein wenig auf und vereinfachen uns die Lösung.)

## b) Problemabstraktion:

- *Gegeben*: ( $n, k \in \mathbb{N}$ )
  1. Eine endliche Menge von Bekannten  $B = \{b_1, \dots, b_n\}$  (hoffentlich mit  $B \neq \emptyset$  😊).
  2. Eine endliche Menge an Bedingungen  $C = \{c_1, \dots, c_k\}$ .

# Algorithmenbau a) & b)

## a) Problemspezifikation:

- *Bekanntenliste*: eine Menge von Personen (z.B. als Zeichenketten der Namen).
- *Einladungsliste*: eine Teilmenge der Menge an bekannten Personen.
- *Möglichst viele*: maximale Größe der Menge eingeladener Bekannter.
- *Bedingung*: eine eindeutig überprüfbare Aussage (hier über die Einladungsliste).
- *Bedingungsliste*: eine Menge an Bedingungen.

(Warum Mengen und nicht mehr Listen? Für jede Liste können wir Argumentieren, dass Dopplungen unsinnig sind: Doppelte Einladungen sind ebenso sinnfrei wie doppelte Bedingungen. Damit lockern wir so das Problem ein wenig auf und vereinfachen uns die Lösung.)

## b) Problemabstraktion:

- *Gegeben*:  $(n, k \in \mathbb{N})$ 
  1. Eine endliche Menge von Bekannten  $B = \{b_1, \dots, b_n\}$  (hoffentlich mit  $B \neq \emptyset$  😊).
  2. Eine endliche Menge an Bedingungen  $C = \{c_1, \dots, c_k\}$ .
- *Gesucht*: Eine Menge einzuladender Bekannten  $E \subseteq B$ , die alle Bedingungen  $c \in C$  erfüllt, mit maximalem  $|E|$ .

# Algorithmenbau c) & d)

c) **Algorithmenentwurf:** (Wir probieren einfach jede Konstellation)

# Algorithmenbau c) & d)

c) **Algorithmenentwurf:** (Wir probieren einfach jede Konstellation)

(1) Setze  $\text{Max} = \emptyset$ .

# Algorithmenbau c) & d)

c) **Algorithmenentwurf:** (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):



# Algorithmenbau c) & d)

c) **Algorithmenentwurf:** (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn ( $|E| > |\text{Max}|$ ):

# Algorithmenbau c) & d)

c) **Algorithmenentwurf:** (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :

# Algorithmenbau c) & d)

c) **Algorithmenentwurf:** (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
    - Für alle  $c \in C$ :
      - Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).

# Algorithmenbau c) & d)

c) **Algorithmenentwurf:** (Wir probieren einfach jede Konstellation)

(1) Setze  $\text{Max} = \emptyset$ .

(2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):

Wenn ( $|E| > |\text{Max}|$ ):

Für alle  $c \in C$ :

Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).

Setze  $\text{Max} = E$ .

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
    - Für alle  $c \in C$ :
      - Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn (E erfüllt nicht c): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist Max.

„E erfüllt nicht c“ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „X nur, wenn Y“:  
c scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „X nicht, wenn Y“:  
c scheitert, wenn  $X \in E$  und  $Y \in E$ .

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn (E erfüllt nicht c): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist Max.

„E erfüllt nicht c“ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „X nur, wenn Y“:
  - c scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „X nicht, wenn Y“:
  - c scheitert, wenn  $X \in E$  und  $Y \in E$ .

## d) Korrektheitsnachweis, Verifikation:

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

„ $E$  erfüllt nicht  $c$ “ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „ $X$  nur, wenn  $Y$ “:  
 $c$  scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „ $X$  nicht, wenn  $Y$ “:  
 $c$  scheitert, wenn  $X \in E$  und  $Y \in E$ .

## d) Korrektheitsnachweis, Verifikation:

1. Vollständige Induktion, oder:



# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn (E erfüllt nicht c): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist Max.

„E erfüllt nicht c“ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „X nur, wenn Y“:  
c scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „X nicht, wenn Y“:  
c scheitert, wenn  $X \in E$  und  $Y \in E$ .

## d) Korrektheitsnachweis, Verifikation:

1. Vollständige Induktion, oder:
2. „Textbasiert“.

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn (E erfüllt nicht c): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist Max.

„E erfüllt nicht c“ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „X nur, wenn Y“:  
c scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „X nicht, wenn Y“:  
c scheitert, wenn  $X \in E$  und  $Y \in E$ .

## d) Korrektheitsnachweis, Verifikation:

1. Vollständige Induktion, oder:
2. „Textbasiert“. B und C sind endliche und konstante Mengen (damit ist auch  $\mathcal{P}(B)$  endlich und konstant).

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn (E erfüllt nicht c): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist Max.

„E erfüllt nicht c“ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „X nur, wenn Y“:  
c scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „X nicht, wenn Y“:  
c scheitert, wenn  $X \in E$  und  $Y \in E$ .

## d) Korrektheitsnachweis, Verifikation:

1. Vollständige Induktion, oder:
2. „Textbasiert“. B und C sind endliche und konstante Mengen (damit ist auch  $\mathcal{P}(B)$  endlich und konstant). Daraus werden auch die Schleifen nur endlich oft durchlaufen. Der Algorithmus *terminiert* somit.

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn (E erfüllt nicht c): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist Max.

„E erfüllt nicht c“ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „X nur, wenn Y“:  
c scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „X nicht, wenn Y“:  
c scheitert, wenn  $X \in E$  und  $Y \in E$ .

## d) Korrektheitsnachweis, Verifikation:

1. Vollständige Induktion, oder:
2. „Textbasiert“. B und C sind endliche und konstante Mengen (damit ist auch  $\mathcal{P}(B)$  endlich und konstant). Daraus werden auch die Schleifen nur endlich oft durchlaufen. Der Algorithmus *terminiert* somit.  
Zudem ist er *partiell korrekt*, in jedem Durchlauf wird Max genau dann geändert, wenn E größer ist *und* alle  $c \in C$  erfüllt.

# Algorithmenbau c) & d)

## c) Algorithmenentwurf: (Wir probieren einfach jede Konstellation)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
  - Für alle  $c \in C$ :
  - Wenn (E erfüllt nicht c): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist Max.

„E erfüllt nicht c“ können wir unterschiedlichst testen. Ganz naiv genügt hier ein Test abhängig von der Form:

- (1) „X nur, wenn Y“:  
c scheitert, wenn  $X \in E$  aber  $Y \notin E$ .
- (2) „X nicht, wenn Y“:  
c scheitert, wenn  $X \in E$  und  $Y \in E$ .

## d) Korrektheitsnachweis, Verifikation:

1. Vollständige Induktion, oder:
2. „Textbasiert“. B und C sind endliche und konstante Mengen (damit ist auch  $\mathcal{P}(B)$  endlich und konstant). Daraus werden auch die Schleifen nur endlich oft durchlaufen. Der Algorithmus *terminiert* somit.  
Zudem ist er *partiell korrekt*, in jedem Durchlauf wird Max genau dann geändert, wenn E größer ist *und* alle  $c \in C$  erfüllt. Da alle möglichen Konstellationen  $E \subseteq B$  geprüft werden, findet sich so das E mit maximaler Kardinalität.

# Algorithmenbau e)

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):
  - Wenn ( $|E| > |\text{Max}|$ ):
    - Für alle  $c \in C$ :
      - Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).
  - Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn  $(|E| > |\text{Max}|)$ :  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .



# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn  $(|E| > |\text{Max}|)$ :  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

e) Aufwandsanalyse:

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn ( $|E| > |\text{Max}|$ ):  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn ( $|E| > |\text{Max}|$ ):  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn  $(|E| > |\text{Max}|)$ :  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die erste Zuweisung 1 ist exakt eine Elementaroperation.

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn  $(|E| > |\text{Max}|)$ :  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die erste Zuweisung **1** ist exakt eine Elementaroperation.
  - Die äußere Schleife in **2** wird  $2^{|B|}$  mal durchlaufen.

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn  $(|E| > |\text{Max}|)$ :  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die erste Zuweisung **1** ist exakt eine Elementaroperation.
  - Die äußere Schleife in **2** wird  $2^{|B|}$  mal durchlaufen.
  - Die erste Bedingung bedarf 3 Elementaroperationen: 2 Kardinalitätsberechnungen, 1 Vergleich.

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn  $(|E| > |\text{Max}|)$ :  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die erste Zuweisung **1** ist exakt eine Elementaroperation.
  - Die äußere Schleife in **2** wird  $2^{|B|}$  mal durchlaufen.
  - Die erste Bedingung bedarf 3 Elementaroperationen: 2 Kardinalitätsberechnungen, 1 Vergleich.
  - Die innere Schleife wird (bis zu)  $|C|$  mal durchlaufen.

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn  $(|E| > |\text{Max}|)$ :  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die erste Zuweisung **1** ist exakt eine Elementaroperation.
  - Die äußere Schleife in **2** wird  $2^{|B|}$  mal durchlaufen.
  - Die erste Bedingung bedarf 3 Elementaroperationen: 2 Kardinalitätsberechnungen, 1 Vergleich.
  - Die innere Schleife wird (bis zu)  $|C|$  mal durchlaufen.
  - Die innere Schleife enthält eine Elementaroperation.



# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn ( $|E| > |\text{Max}|$ ):  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die erste Zuweisung **1** ist exakt eine Elementaroperation.
  - Die äußere Schleife in **2** wird  $2^{|B|}$  mal durchlaufen.
  - Die erste Bedingung bedarf 3 Elementaroperationen: 2 Kardinalitätsberechnungen, 1 Vergleich.
  - Die innere Schleife wird (bis zu)  $|C|$  mal durchlaufen.
  - Die innere Schleife enthält eine Elementaroperation.
  - Die äußere Schleife enthält weiter noch eine Elementaroperation ( $\text{Max} = E$ ).

# Algorithmenbau e)

- (1) Setze  $\text{Max} = \emptyset$ .
- (2) Für alle  $E \in \mathcal{P}(B)$  (alternativ:  $E \subseteq B$ ):  
    Wenn ( $|E| > |\text{Max}|$ ):  
        Für alle  $c \in C$ :  
            Wenn ( $E$  erfüllt nicht  $c$ ): Weiter über (2).  
        Setze  $\text{Max} = E$ .
- (3) Ergebnis ist  $\text{Max}$ .

Aus der Mathematik ist bekannt, dass für eine endliche Menge  $|\mathcal{P}(X)| = 2^{|X|}$ .

## e) Aufwandsanalyse:

1. Tabellarisch, siehe Vorlesung, oder:
2. „Textbasiert“. Wir betrachten Stückweise:
  - Die erste Zuweisung **1** ist exakt eine Elementaroperation.
  - Die äußere Schleife in **2** wird  $2^{|B|}$  mal durchlaufen.
  - Die erste Bedingung bedarf 3 Elementaroperationen: 2 Kardinalitätsberechnungen, 1 Vergleich.
  - Die innere Schleife wird (bis zu)  $|C|$  mal durchlaufen.
  - Die innere Schleife enthält eine Elementaroperation.
  - Die äußere Schleife enthält weiter noch eine Elementaroperation ( $\text{Max} = E$ ).
3. Damit erhalten wir:  $1 + 2^{|B|} \cdot (3 + |C| \cdot (1) + 1) = 1 + 2^{|B|} \cdot (4 + |C|)$



