1

# Mathe. Haha. Kann ich.

## Tutorium Unostasio

Florian Sihler ○ KW 44

# All hail the space!

Lehrfeld? Lärfeld? Learfeld? Nix?

# 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

- 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)

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

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

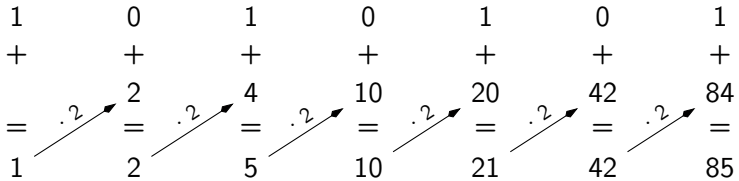
- $1010101_{(2)} = (((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1 = 85_{(10)}$
- $12345_{(8)} = ((((1 \cdot 8 + 2) \cdot 8 + 3) \cdot 8 + 4) \cdot 8 + 5) = 5349_{(10)}$
- Hexadezimal:

$1_{(16)} \mathrel{\widehat{=}} 1_{(10)}$
$2_{(16)} \mathrel{\widehat{=}} 2_{(10)}$
$3_{(16)} \mathrel{\widehat{=}} 3_{(10)}$
$4_{(16)} \mathrel{\widehat{=}} 4_{(10)}$
$5_{(16)} \mathrel{\widehat{=}} 5_{(10)}$
$6_{(16)} \mathrel{\widehat{=}} 6_{(10)}$
$7_{(16)} \mathrel{\widehat{=}} 7_{(10)}$
$8_{(16)} \mathrel{\widehat{=}} 8_{(10)}$
$9_{(16)} \mathrel{\widehat{=}} 9_{(10)}$
$A_{(16)} \mathrel{\widehat{=}} 10_{(10)}$
$B_{(16)} \mathrel{\widehat{=}} 11_{(10)}$
$C_{(16)} \mathrel{\widehat{=}} 12_{(10)}$
$D_{(16)} \mathrel{\widehat{=}} 13_{(10)}$
$E_{(16)} \mathrel{\widehat{=}} 14_{(10)}$
$F_{(16)} \mathrel{\widehat{=}} 15_{(10)}$

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

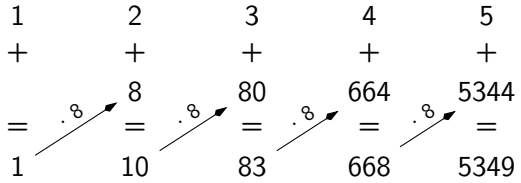
# Übungsblatt 1 - Aufgabe 1 a)

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



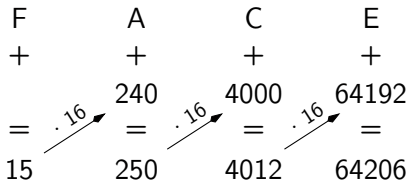
# Übungsblatt 1 - Aufgabe 1 a)

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



# Übungsblatt 1 - Aufgabe 1 a)

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



# Übungsblatt 1 - Aufgabe 1 b) Vorbemerkung

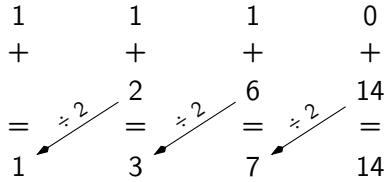
- $14_{(10)}$ zur Basis 2. Eine Vorbemerkung:

$$\begin{aligned} 14 \div 2 &= 7 & 14 \bmod 2 &= 0 \quad (\leftarrow \text{LSB}) \\ 7 \div 2 &= 3 & 7 \bmod 2 &= 1 \\ 3 \div 2 &= 1 & 3 \bmod 2 &= 1 \\ 1 \div 2 &= 0 & 1 \bmod 2 &= 1 \quad (\leftarrow \text{MSB}) \end{aligned}$$

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



# 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

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

▼

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.

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)

a) **Problemspezifikation:**

- *Tagespreis:* positive reelle Zahl. Dargestellt mit zwei Nachkommastellen.
- *Tagespreisliste:* chronologisch sortierte Liste an Tagespreisen $[p_1, \ldots, p_n]$ 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 $[p_1, \ldots, p_n]$ positiver reeller Zahlen. Annahme: $n > 1$ („letzte Monate“).
- *Gesucht:* Positive reelle Zahl $m = 100 \cdot \max_{1 \leq i < n} \frac{p_{i+1} - p_i}{p_i}$ ($X\,\% \mathrel{\widehat{=}} \frac{X}{100}$).

# Übungsblatt 1 - Aufgabe 2 c) & d)

c) Algorithmenentwurf:

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

(2) Setze $i = 1$.

(3) Solange $(i < n)$, wiederhole:

Setze test $= (p_{i+1} - p_i)/p_i$.

Wenn test $> \Delta_{max}$: Setze $\Delta_{max} =$ 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 $[p_1, \ldots, p_{i+1}]$ im i-ten Durchlauf. Nach $n - 1$ Durchläufen: $[p_1, \ldots, p_n]$. Basisfall mit $i = 2$, für Schritt $i \to i + 1$.

# Übungsblatt 1 - Aufgabe 2

Notizen

- 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.)

# Übungsblatt 1 - Aufgabe 2 d)



(1) Setze $\Delta_{max} = 0$.
(2) Setze $i = 1$.
(3) Solange ($i < n$), wiederhole:
    Setze $\texttt{test} = (p_{i+1} - p_i)/p_i$.
    Wenn $\texttt{test} > \Delta_{max}$: Setze $\Delta_{max} = \texttt{test}$.
    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 $p_1, \ldots, p_n$ 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. $\texttt{test} > \Delta_{max}$: Durch die Bedingung wird $\Delta_{max}$ aktualisiert. Es gilt: $\Delta_{max} = \texttt{test} \geq \max_{1 \leq j \leq i+1}((p_{j+1} - p_j)/p_j)$.
2. $\texttt{test} \leq \Delta_{max}$: Das neue Glied durch $i + 1$ ist kein neues Maximum. Es gilt: $\Delta_{max} \geq \max_{1 \leq j \leq i+1}((p_{j+1} - p_j)/p_j) \geq \texttt{test}$.

Nach $n - 1$ Schritten durch die Begrenzung $i < n$, wurden alle $p_1, \ldots, p_n$ 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 $\texttt{test} = {}^{(p_{i+1} - p_i)}/_{p_i}$.
    Wenn $\texttt{test} > \Delta_{max}$: Setze $\Delta_{max} = \texttt{test}$.
    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

- 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:
  - $s$ 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

- 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 $2^{-1}2^{-2}2^{-3}\ldots$): $7.25_{(10)} = 111.01$
  3. Normalisiere Mantisse: $111.01 = 1.1101 \cdot 2^2 \Rightarrow M = 1101$
  4. Berechne Exponent: $e = 2 \overset{+B=1}{\Rightarrow} E = 3_{(10)} = 11_{(2)}$
  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 ☺).
  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

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

- 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 $|\,e\,| = 2$): $11_{(2)} \widehat{=} 3_{(10)}$ 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: $111.01_{(2)} \widehat{=} 7 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 7.25_{(10)}$
  5. Füge Vorzeichen an (hier positiv): 7.25, ferddisch! (und schdimmd, 😊)

# Das IEEE 754 Format - Rundungsproblematik

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

```java
class RoundingProblem {
    public static void main(String[] args) {
        double money = 0.1D;
        System.out.println(money);
        System.out.println(1.0 + money - 1.0);
    }
}
```

- Erwartung: 0.1 und 0.1

# Das IEEE 754 Format - Rundungsproblematik

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

```
0.1
0.10000000000000009
```

- Warum?
  $0.1_{(10)} \widehat{=} 0.00011001100110011\ldots_{(2)} \neq 0.1_{(10)}$ da Mantisse begrenzt.

# 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 Einlandungsliste berechnet, die möglichst viele Personen einlädt.

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

# 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 **Einlandungsliste berechnet**, die möglichst viele Personen einlädt.

Bekanntenliste
$(b_1, \ldots, b_n)$
Bedingungsliste
$(c_1, \ldots, c_k)$

▼

Einladeliste
$(e_1, \ldots, e_j)$

# 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 Einlandungsliste berechnet, die **möglichst viele** Personen einlädt.

Begriffe:

Bekanntenliste
Bedingungsliste
Einladungsliste
möglichst viele

# 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 $B = \{b_1, \ldots, b_n\}$ (hoffentlich mit $B \neq \emptyset$ 😊).
  2. Eine endliche Menge an Bedingungen $C = \{c_1, \ldots, c_k\}$.
- *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)

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

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

(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)

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

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

(3) Ergebnis ist 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 ($\mathrm{Max} = E$).
3. Damit erhalten wir: $1 + 2^{|B|} \cdot (3 + |C| \cdot (1) + 1) = 1 + 2^{|B|} \cdot (4 + |C|)$