2. Algorithmen

Rechneranwendung braucht
(i) Verfahren zur Lösung ➔ Begriff „Algorithmus“
(ii) in dem Rechner verständl. Form ➔ Begriff „PS“

2.1. Begriff des Algorithmus

Beispiel 2.1: Suche des maximalen Elementes

1. Eingabe der 1. Zahl.
2. Setze MAX gleich Wert der 1. Zahl.
3. Für Zahl an i-ter Stelle, wobei i nacheinander gleich
   2, 3, ... bis 10 000, führe aus:
   3.1. Eingabe der i-ten Zahl.
   3.2. Wenn Wert der i-ten Zahl > MAX
   dann setze neu MAX gleich Wert der i-ten Zahl
   sonst tue nichts.
4. Ausgabe von MAX.
5. Stopp.

➔folgende
Def. 2.1. (Algorithmus)

Algorithmus := *Menge von Regeln* für ein Verfahren,
um *aus gegebenen Eingabegrößen* bestimmte *Ausgabegrößen herzuleiten*, wobei folgende *Bedingungen* erfüllt sein müssen:
(1) FINITHEIT der Beschreibung,
(2) EFFEKTIVITÄT,
(3) TERMINIERTHEIT,
(4) DETERMINIERTHEIT.

- weitere Eigenschaften:
  (A) A. löst ***Klasse*** von Aufgabenstellungen
  (B) Sequentieller Ablauf
  (C) Formulierung der Regeln für Ausführenden – weiterer Abstraktionsschritt

- Prozeß ↔ Prozessor

- Ist vorgesehener Prozessor ein Rechner ➔
  - A. in Form eines ***Programmes***, das
  - in einer ***Progr.-sprache*** niedergeschrieben.
  - Entwurf eines A. → ***Algorithmieren***
  - Entwurf eines Pr. → ***Programmieren***

- grundlegende Rolle von A.

<u>Zusammenfassung:</u>

Für Informatiker sind A. von grundlegenderer Bedeutung als PS-en oder auch Rechner.
Eine PS ist einfach ein brauchbares Mittel, um einen A. auszudrücken;
ein Rechner ist einfach ein Prozessor, um einen A. auszuführen.
Beide sind Mittel zum Zweck.

2.2. Bausteine und Darstellung von Algorithmen

- elementare Operationen
- zyklische Wiederholungen
- Entscheidungen

2.2.1. Darstellungsformen von Algorithmen

1. menschliche Umgangssprache

2. Pseudokode

Bsp. 2.2: Multiplikation zweier natürlicher Zahlen

Geg.: m, n – natürliche Zahlen;
Ges.: r = m * n

MULT1(↓m, ↓n, ↑r);

(1) r ← 0;
(2) wenn (m = 0 oder n = 0) dann HALT;
(3) r ← r + m;
(4) n ← n–1;
(5) wenn n = 0 dann HALT
sonst weiter bei (3).

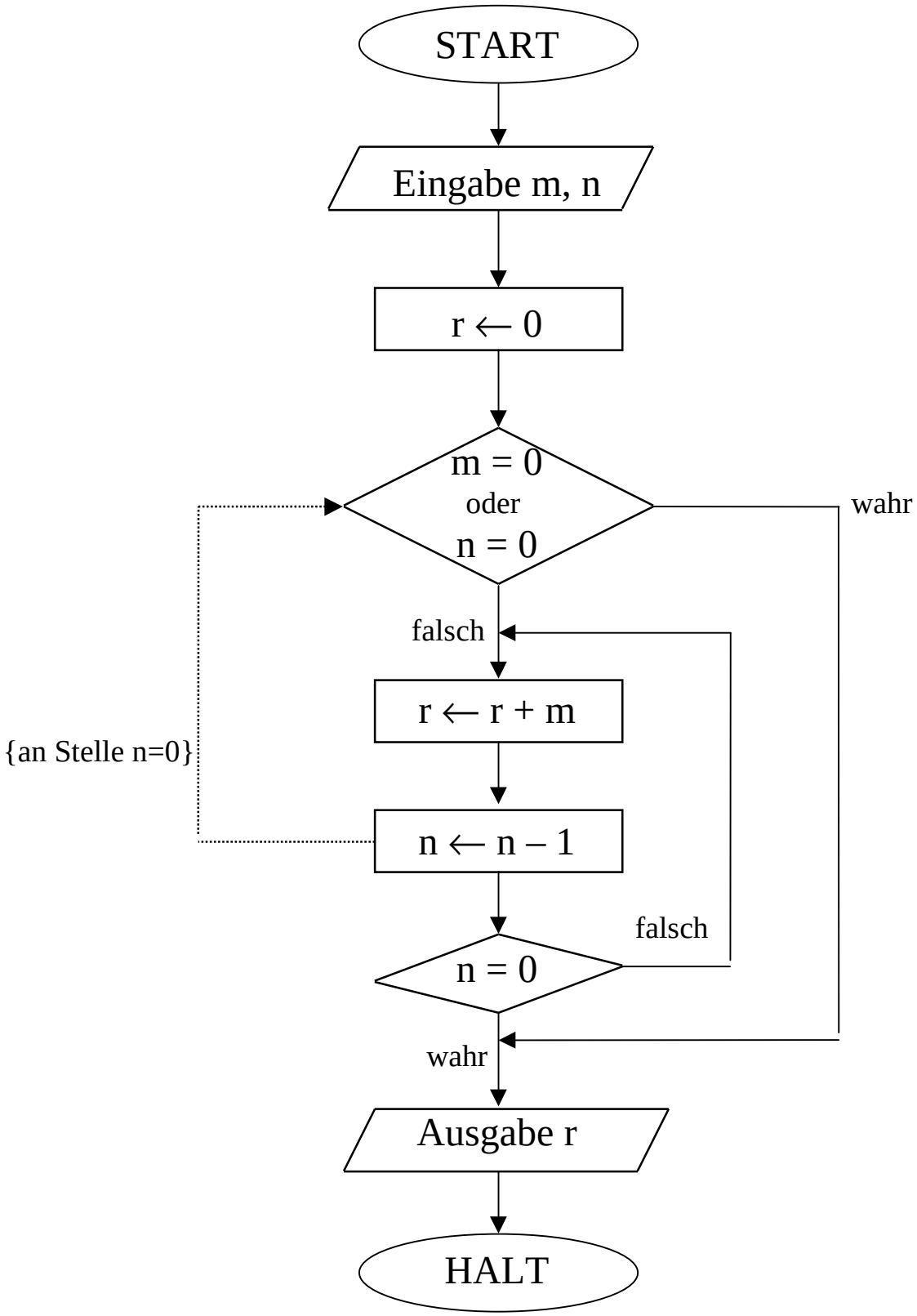
* Bem.: Vorgriff auf Begriff ***Datenobjekt***

DO = (Name, Typ, Wert)

Typ: T = (W, O)

3. grafische Darstellung

Flußdiagramm für MULT1(↓ m, ↓ n, ↑ r)



2.2.2. Einfache Anweisungen

(A) Eingabeanweisung

	lies(liste)	Eingabe(liste)
Bsp.:	lies(m, n)	Eingabe(m, n)

(B) Ausgabeanweisung

	schreib(liste)	Ausgabe(liste)
Bsp.:	schreib(r)	Ausgabe(r)

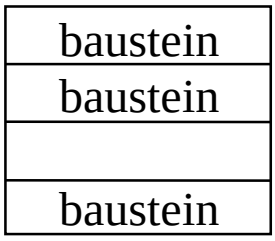
(C) Wertzuweisung

	variable ← ausdruck	variable := ausdruck
Bsp.:	r ← 0	r := 0

2.2.3. Folge (Sequenz)

= unverzweigte Folge von Bausteinen
(die einer nach dem anderen auszuführen sind)

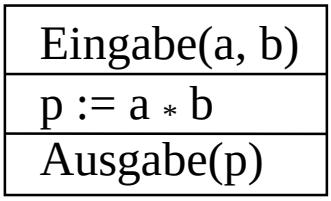
```
BEGINN
    baustein;
    baustein;
    ...
    baustein
ENDE
```



Bem.:
BEGINN und ENDE – syntaktische Klammern.
Klammern die zur Folge gehörenden Bausteine bzw. geben die Grenzen der Sequenz an.

Bsp.: Produkt p zweier Werte a und b gesucht.

```
BEGINN
    lies(a, b);
    p ← a * b;
    schreib(p)
ENDE
```



Aber: Ein A., der nur aus Folge, sehr unflexibel.
Z.B. wenn A. für Zubereitung 1 Tasse Kaffee und nun 2 benötigt?

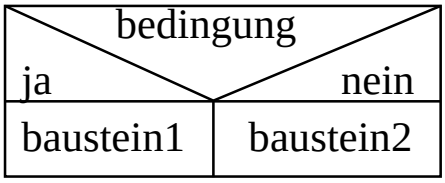
==> Kombination von Bausteinen als Folge – sehr primitive A.-struktur.

2.2.4. Alternative (Auswahl, Selektion)

- oft Auswahl bezgl. Ausführung von Anweisungen treffen (vgl. MULT1)
  häufigste Fall: Auswahl zwischen zwei Bausteinen

- allg. Form:

```
WENN   bedingung   {erfüllt}
    DANN   baustein1
    SONST  baustein2
```

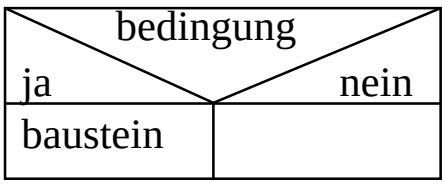


- Bez.:

  Eine Konstruktion, bei der in Abh. davon, ob eine Bed. erfüllt ist (ja) oder nicht (nein), genau 1 von 2 Algorithmenbausteinen ausgeführt wird, heißt (vollständige) ***Alternative***.

  Steht bei „nein“ bzw. SONST ein Leerbaustein ==> ***unvollständige Alternative***:

```
WENN   bedingung
    DANN   baustein
```



z.B. in MULT1 ==> WENN (m = 0 oder n = 0)
DANN Halt.

Bem.:

a) Prozessor muß bedingung ***und*** baustein verstehen.
b) Unvollst. Alternative ist Spezialfall der vollständigen.
c) Einrückschreibweise zur besseren Übersicht

■Beispiel 2.3:

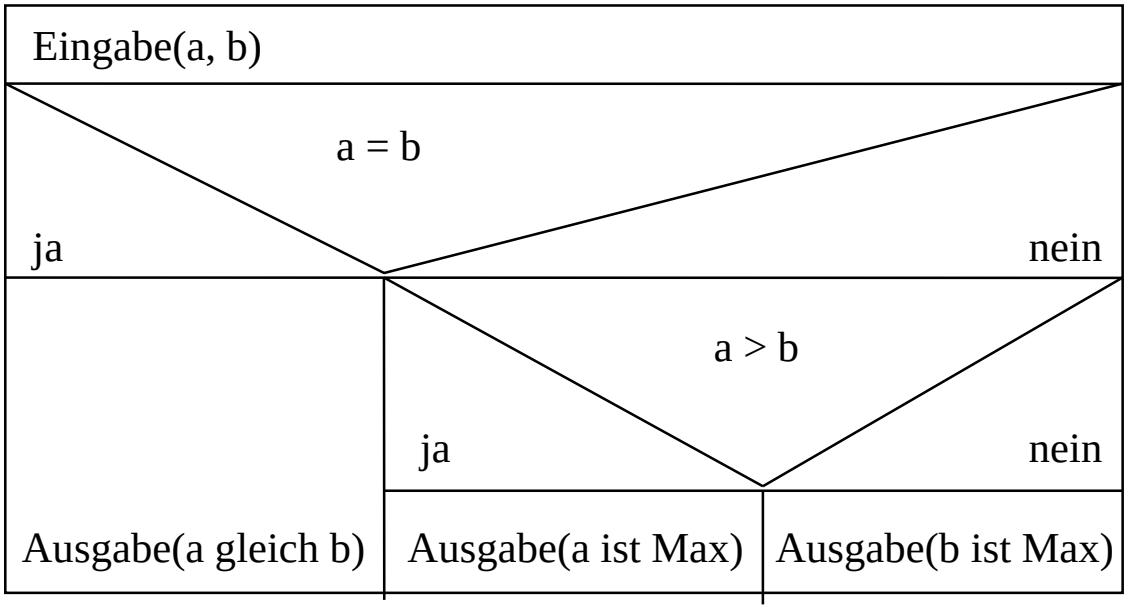
A. zur Bestimmung der größeren von zwei Zahlen a und b.

```
BEGINN
    lies(a, b);
    WENN   a = b
        DANN   schreib(a gleich b)
        SONST  WENN   a > b
                    DANN   schreib(a ist Max)
                    SONST  schreib(b ist Max)
ENDE
```

Bem.:

- Zweifache Auswahl – die 2. liegt geschachtelt in 1. Schachteltiefe von Rechner abhängig.

- Ohne Einrückschreibweise schwer zu verstehen.



■ Manchmal mehr als 2 alternative Bausteine ➔ folg. Aufgabe:

Für vorgegebene Anzahl Studierender ist in einem Fach die Anzahl der verschiedenen Noten 1 bis 5 zu ermitteln.

==> im A. dann folg. Teil:

```
WENN   zensur = 1    DANN    eins ← eins + 1;
WENN   zensur = 2    DANN   zwei ← zwei + 1;
WENN   zensur = 3    DANN    drei ← drei + 1;
WENN   zensur = 4    DANN    vier ← vier + 1;
WENN   zensur = 5    DANN  fuenf ← fuenf + 1;
```

auch als Folge verschachtelter vollst. Alternativen ➔

```
WENN   zensur = 1
   DANN   eins ← eins + 1
   SONST  WENN   zensur = 2
                 DANN   zwei ← zwei + 1
                 SONST  WENN   zensur = 3
                               DANN   drei ← drei + 1
                               SONST  WENN   zensur = 4
                                             DANN   vier ← vier + 1
                                             SONST  fuenf ← fuenf + 1
```

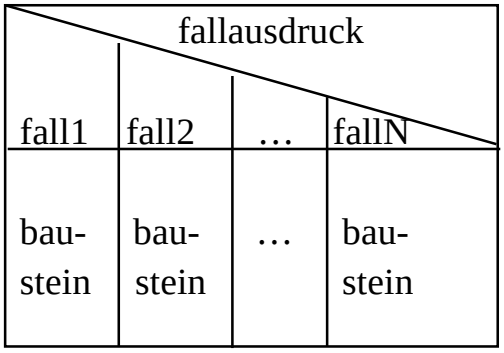
■Bem.:

Hier ist Aufwand für Auswertung der Bedingungen kleiner, dafür aber unübersichtlicher.

➔ spezieller Baustein

■Baustein ***Fallunterscheidung***:

```
FALL   fallausdruck
VON
        fall1: baustein;
        fall2: baustein;
        …
        fallN: baustein
ENDE
```



Bsp.:

```
FALL      zensur
VON
        1:    eins ←   eins + 1;
        2:    zwei ←   zwei + 1;
        3:    drei ←   drei + 1;
        4:    vier ←   vier + 1;
        5:   fuenf ←  fuenf + 1
ENDE
```

Bem.: Auf diese Art auch Menütechnik realisierbar. ➔

```
lies(zeichen);
FALL zeichen
VON
        E, e:  Editor aktivieren;
        C, c: Compiler aktivieren;
        R, r: Run eines Algorithmus ausführen
ENDE
```

■**Ablauf** der Fallunterscheidung:

Besteht aus zu bewertendem ***Fallausdruck*** und ∀ langen Folge von Fall–***Marken***, auf die jeweils **<u>ein</u>** Baustein folgt.
Wert des Fallausdrucks nacheinander mit den Marken verglichen. Stimmen beide überein, so wird
(a) der hinter der Marke stehende Baustein ausgeführt und
(b) die Fallunterscheidung verlassen (die restl. Marken übersprungen).
Der zu bewert. Fallausdruck heiß ***Selektor***.
Auch mehrere Marken oder Bereichsangaben möglich.

2.2.5. Wiederholung (Iteration)

■folg. Aufgabe:

Geg.: Liste von Namen und Adressen; Name einer Person;
Ges.: Adresse dieser Person;

Vor.: Gesuchte Name ist in der Liste!

■mögliche Lösung:

```
lies(ersten Namen der Liste);
WENN dieser Name gleich geg. Name
   DANN  schreib(zugehörige Adresse)        ⎫ b1
   SONST lies(nächsten Namen);              ⎭
      WENN dieser Name gleich geg. Name
         DANN  schreib(zugehörige Adresse)  ⎫ b1
         SONST lies(nächsten Namen);        ⎭
            WENN …
```

➔ **Problem: Wann aufhören?**

⇨Folge und Alternative allein unzureichend zur Wiedergabe von A., deren „Länge“ mit den Umständen variiert.

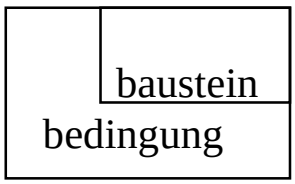
■Konstrukt zur beliebig häufigen Wdh. von A.-teilen:

Führen die Worte WIEDERHOLE und BIS ein ➔

```
lies(ersten Namen der Liste);
WIEDERHOLE
      WENN dieser Name gleich geg. Name
         DANN  schreib(zugehörige Adresse)  ⎫ b1
         SONST lies(nächsten Namen)         ⎭
BIS  geg. Name gefunden oder Liste durchgesehen;
```

■eine Form der Wiederholung, die allgemein darstellbar als

WIEDERHOLE
baustein
BIS bedingung;



Bedeutung:

Jener Teil des A. (baustein), der zwischen den Worten WIEDERHOLE und BIS steht, ist wiederholt auszuführen, bis die Bedingung erfüllt ist, welche hinter dem BIS steht.

Bezeichnungen:

Schleife; Schleifenkörper (-rumpf); Abbruchbedingung

■Bem.:

Bedeutung der Schleife – Prozeß mit *unbestimmter* Dauer durch A. *endlicher* Länge beschreibbar!

⇨ Verantwortung: *korrekte Abbruchbedingung*!!!
{ einer der häufigsten Fehler beim A.-entwurf }

■Bem.:

Schleifenkörper mind. 1× durch **ehe** Abbruchbed. geprüft ==>

Nichtabweisschleife

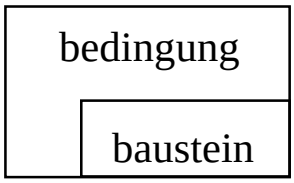
Bedeutung dieser Tatsache „ehe“ am Bsp. „Namensliste“ ==>
Was, wenn Liste nur einen Namen enthält oder der geg. Name nicht in der Liste? ==>

■**Bedarf** für Abbruchbedingung **vor** Schleifenkörper!
➔ für Bsp. Namensliste: {Liste nicht leer}

```
lies(1. Namen der Liste);
WENN dieser Name gleich geg. Name
   DANN schreib(zugehör. Adresse);
SOLANGE geg. Name nicht gefunden und Liste nicht
                durchgesehen
FUEHRE AUS
      BEGINN
            lies(nächsten Namen);
            WENN dieser Name gleich geg. Name
               DANN schreib(zugehörige Adresse)
      ENDE;
```

■allg. Form:

```
SOLANGE      bedingung
FUEHRE AUS   baustein;
```



■Bez.:

Schleifenkörper – alles nach FUEHRE AUS;
Abbruchbedingung – **vor** Schleifenkörper zw. SOLANGE und FUEHRE AUS;
Abweisschleife – Test der Abbruchbed. **vor** erstmaligem Durchlaufen des Schleifenkörpers

■Bem.:

(1) Nichtabweisschleife nur anwendbar, wenn klar, daß mind. 1 Durchlauf des Schleifenkörpers.
(2) Sind beide Arten zulässig, so ineinander überführbar.

■ Zählschleife:

```
FUER   i:=a   BIS   e
    FUEHRE AUS   baustein;
```

FUER i:=a BIS e	
	baustein

i – ***Zähl-*** oder ***Laufvariable***
a – Anfangswert;
e – Endwert:

➔ **Wirkung**:

1. Eintritt in Zählschleife bei a ≤ e. Abarbeitung von baustein für i = a.
2. Es wird i := i+1 berechnet.
3. Bleibt i in [a, e], so Abarbeitung von baustein und Fortsetzung mit 2.
   Andernfalls Austritt aus Schleife.

■ Beispiel 2.4: Summe von N reellen Zahlenwerten

Aufgabe: A. für die Berechnung der Summe s von N reellen Zahlenwerten a1 bis aN.

Lösung: vorbereitende Betrachtung ==>
s = a1+a2+…+aN = (a1+a2+…+aN-1)+aN
s(n) = a1+a2+…+an für n = 0…N mit s(0) = 0 ➔

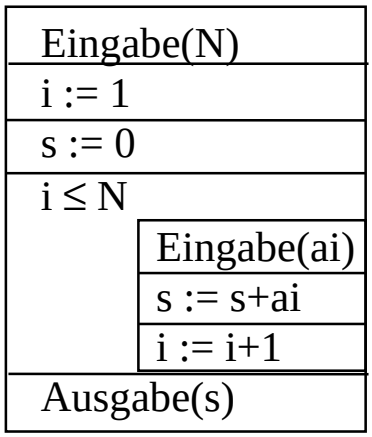
s(n) = s(n-1) + an ➔ folg. A.

```
lies(N);
s := 0;
FUER i := 1 BIS N
      FUEHRE AUS
            BEGINN
                  lies(ai);
                  s := s+ai
            ENDE
schreib(s);
```

Eingabe(N)	
s := 0	
FUER i := 1 BIS N	
	lies(ai)
	s := s+ai
Ausgabe(s)	

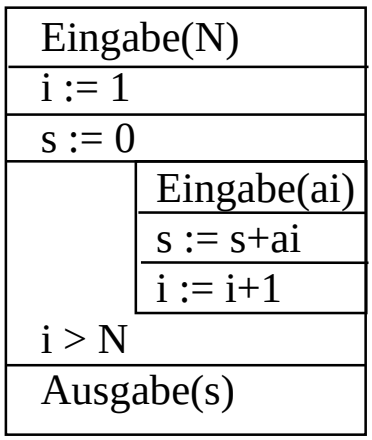
■ als A. mit ***Abweisschleife***:

```
lies(N);
i := 1;
s := 0;
SOLANGE   i ≤ N
  FUEHRE AUS
      BEGINN
          lies(ai);
          s := s+ai;
          i := i+1
      ENDE
schreib(s);
```



■ als A. mit ***Nichtabweisschleife***:

```
lies(N);
i := 1;
s := 0;
WIEDERHOLE
          lies(ai);
          s := s+ai;
          i := i+1
BIS   i > N;
schreib(s);
```



HAUSAUFGABE: (1) A. zur Multiplikation von N Zahlen.
(2) A. zur Best. des Max. von N Zahlen.

■ **Bem.:**

1. Bausteine Folge, Alternative und Wiederholung ausreichend!
2. Englische Codewörter ==> **fast** PS!

2.3. Spezifikation von Algorithmen

■ Algorithmierung ==> versch. Schritte:

1. Beschreibung des Problems / der Aufgabe
2. Entwurf
3. Test
4. Komplexitätsaussagen

■ Spezifikation ::= Festlegung/Def. der Aufgabenstellung und Eigenschaften des zu entwerfenden A.

formal $f: E \rightarrow A$

■ Warum bzw. wozu? ==> folg.

Beispiel 2.5: Parkplatz

{Klaeren, §2.1}

Auf Parkplatz stehen PKW und Motorräder ohne Beiwagen, insgesamt n Fahrzeuge mit m Rädern.
Aufgabenstellung: Bestimme die Anzahl P von PKW´s.

- (vorbereit.?) Überlegungen (Problemanalyse):

(1) P(n, m) – Anzahl PKW bei Parameterpaar (n, m)
M(n, m) – Anzahl Motorräder bei … (n, m)

Klar: $P(n, m) + M(n, m) = n$
$4 * P + 2 * M = m$

➔ Lösung: $\boxed{P = (m - 2n) / 2}$ (*)

Frage: Problem gelöst?

(2) „Test“–Rechnung: n = 3, m = 9

P(3, 9) = 1.5 !!!

2. „Test“–Rechnung: n = 5, m = 2

P(5, 2) = –4 !!!

3. „Test“–Rechnung: n = 2, m = 10

P(2, 10) = 3 → Okay?

M(2, 10) = n – P(2, 10) = –1 ↯

Ursache der Schwierigkeiten:

Abstraktionsschritt von Problembeschreibung zu Parametern n und m ohne problem**typische** Eigenschaften.

⇓

■ Spezifikationsregel: (Klaeren S.24)

Vor der Entwicklung eines A. ist für das Problem eine ***funktionale Spezifikation*** anzufertigen. Diese beschreibt die Menge der gültigen Eingabewerte (**Definitonsbereich**) und die Menge der möglichen Ausgabewerte (**Wertebereich**) mit allen für die Lösung wichtigen Eigenschaften, insbesondere dem funktionalen Zusammenhang zwischen Ein- und Ausgabewerten.

Empfehlung: Zush. zw. Def. und Wertebereich durch sogen. ***Vor-*** und ***Nachbedingungen*** beschreiben.

Vorbedingung: Zustand **vor** Ausführung eines geplanten A., also seine Voraussetzungen (für Anwendung)

Nachbedingung: Zustand **nach** Ausführung des A., seine Leistung

■ Anwendung auf Bsp. 2.5 ==>

folg. Spezifikation:
Zu bestimmen ist die Anzahl p(n, m) von PKW‘s auf einem Parkplatz mit n Fahrzeugen und zusammen m Rädern. Die Fahrzeuge sind nur PKW´s und Motorräder.

Eingabe: m, n ∈ **N**

Vorbedingung: m – gerade; $2n \leq m \leq 4n$

Ausgabe: p(n, m) ::= P ∈ **N**, falls Nachbedingung erfüllbar, sonst „Keine Lösung !“.

Nachbedingung: Für gewisse P, M ∈ **N** gilt

$$P + M = n$$
$$4P + 2M = m$$

■ Bemerkung: A. (Programm) soll i.d.R. ***robust*** sein gegenüber nichtzulässigen Eingaben

⇨ **<u>vor</u>** Start sind nun Vorbedingungen überprüfbar

➔ WENN Vorbedingungen eines A. erfüllt
DANN Abarbeitung
SONST (Fehler-)Meldung;

■ Anwendung auf Bsp. 2.5 → Algorithmus „Parkplatz“:

Zu bestimmen ist die Anzahl p(n, m) der PKW´s auf einem Parkplatz mit n Fahrzeugen und zusammen m Rädern. Die Fahrzeuge sind nur PKW´s und Motorräder.

Eingabe: m, n ∈ **N**

Vorbedingung: m – gerade; $2n \leq m \leq 4n$

Verfahren: Berechne P ::= (m – 2n) / 2

Ausgabe: p(n, m) ::= P ∈ **N**

Nachbedingung: Es gibt M ∈ **N** mit

$$P + M = n$$
$$4P + 2M = m$$

Bemerkung:

Guter Stil – Wdh. der Spezifikation in A.-beschreibung.

■ Bsp. 2.5 → Standard–Objekte aus Mathematik:
allg. Notwendigkeit: Def. ***problemspezifischer Objekte***

Grundregel: | Nur mit solchen Objekten arbeiten, die in ihren Eigenschaften vollständig bestimmt sind. |

⇨ erleichtert a) Formulierung des Problems
b) Kontrolle/Fehlersuche

➔ Begriff des Datenobjektes oder des **Datums:**

Datum ::= (Name, Typ, Wert)

Name: *Grundregel* „Alles was in A. benutzt wird, muß (s)einen Namen (Bezeichner) haben."
Grundregel „Alle Namen sind **vor** ihrer erstmaligen Verwendung zu deklarieren."
Typ: T = (W, O)
Wert: durch Zuweisung; W – Wertemenge

allg. unterteilen wir Daten in Variable und Konstanten

Variablen: * nehmen im A. *beliebige* Werte aus typspezif. Wertebereich an;
* erhalten Wert durch Wertzuweisung oder Eingabe;
Konstanten: * per Def. einmalig ein *unveränderlicher* Wert
* dürfen nur als Operanden in Ausdrücken ihres Typs auftreten;

➔

■Definition 2.2 (Spezifikation)

Eine (funktionale) ***Spezifikation*** besteht aus folgenden Komponenten:

1. Deklarationen von Konstanten, Datentypen, Funktionen, Variablen.
2. Eingabe.
3. Vorbedingung.
4. Ausgabe.
5. Nachbedingung.

Erklärung:

zu 1.: a) *Konstantenbezeichner*
b) Typvereinbarung: *Typbezeichner* (für W) und *Operationenbezeichner* (für Elemente aus O)
c) *Funktionsbezeichner* für zu berechnende Fkt-nen
d) *Variablenbezeichner* mit zugeordneten Datentypen

zu 2.: *Variablenbezeichner* für die *Eingabeparameter*

zu 3.: Menge von Eigenschaften (*Prädikaten*) für Eingaben i.d.R. auch die Bez. der E.–Parameter verwendet

zu 4.: *Variablenbezeichner* für die *Ausgabeparameter*

zu 5.: Menge von Prädikaten für die Leistung des A. i.d.R. treten hier auch die A.–Parameter auf

■Beispiel 2.6: Suche nach einem Objekt

Innerhalb einer endlichen Folge von Elementen ist die Position eines bestimmten Elementes anzugeben. Ist das bestimmte Element nicht in der Folge, soll die Position 0 ausgegeben werden. Dabei wird angenommen, daß die Folge keine zwei identischen Elemente enthält.

⇨ z. B. folgende **Spezifikation:**

Zu bestimmen ist die Position P(a, A) eines Elementes a in einer Folge A. Es gelte P(a, A) = 0 für $a \notin A$.

1. Deklaration:

Datentypen: M – eine Menge;
$A(M)$ – die Menge aller endl. Folgen über M
schreiben Folge A als $A = (a_1, a_2, ..., a_n)$

Funktionen: $P : M \times A(M) \rightarrow \boldsymbol{N}$

2. Eingabe: $A = (a_1, a_2, ..., a_n) \in A(M)$
$a \in M$
es sei $n \geq 1$

3. Vorbedingung: $a_i \neq a_j$ für $i \neq j$, $i, j = 1..n$

4. Ausgabe:

$$P(a, A) = \begin{cases} p & \text{falls } \exists\, p \in \{1,\ldots,n\} \text{ mit } a_p = a \\ 0 & \text{sonst} \end{cases}$$

5. Nachbedingung:

$$(a = a_p \wedge 1 \leq p \leq n) \vee (\forall j \in \{1,...,n\}: a \neq a_j \wedge p = 0)$$

■ Zusammenfassung:

1. Vor Entwicklung eines A. – funktionale Spezifikation, d.h., es sind Ein- und Ausgabebereich sowie der funktionelle Zusammenhang zw. Eingabe- und Ausgabewerten festzulegen.

2. Eine präzise Spezifikation kann als Basis für die weitere Arbeit einen erheblichen Teil des gedanklichen Aufwandes innerhalb des Problemlösungsprozesses ausmachen.

3. Wichtigste Grundregel ist es, dass man nur mit bezüglich ihrer Eigenschaften wohldefinierten Objekten arbeitet.

2.4. Entwurf von Algorithmen

■ ∃ Art Regelwerk für Entwurf

2 Beispiele für allgemeine Fehler beim Algorithmieren:

(1) Der beschriebene Ablauf ist **beinahe** der angestrebte!

(2) **Normalerweise** (!) wird der gewünschte Ablauf erreicht, nur unter bestimmten Umständen nicht, die leider vom Entwickler unbemerkt blieben.

➔ einzige Lehre daraus: **methodisches Vorgehen**

■ **schrittweise Verfeinerung** oder **Top–Down**–Entwurfsmethode

• ***Grundgedanke***:
Auszuführende Prozeß in mehrere Einzelschritte, so daß jeder durch einen A. beschrieben, der weniger umfangreich und einfacher als der für ursprünglichen Prozeß.
Dabei gilt:

(a) Unterprobleme (UP) sind lösbar.
(b) Lösung eines UP i.W. ohne Einfluß auf Lösung anderer UP.
(c) Lösung eines UP – geringeren Umfang als Lös. des Urproblems.
(d) Sind alle UP gelöst, für Gesamtlösung keinen Zusatzaufwand.

Zerlegung mehrfach ➔ folg. einfache **Prinzip**:

1. Start mit **Grobentwurf**.
2. Mehrere **Verfeinerung**sschritte.

Frage: Wann mit Verfeinerung aufhören ?

Antwort: Sobald alle Anweisungen vom Prozessor ausführbar sind!!!

➔ Wissen, welche Schritte Prozessor interpretieren kann.

➔ Kenntnisse zu Möglichkeiten der HW und vor allem der SW in Form von PS-en.

⇨ Dieses Wissen auch für **Richtungsfestlegung** der Verfeinerung!!!

■ Im Zusammenhang mit §2.3 =>

Definition 2.3 (***Top–Down–Entwurfsmethode***)
{Klaeren S.35}

Der Entwurf eines A. nach dem Top–Down–Verfahren (schrittweise Verfeinerung) findet in den folgenden Schritten statt:

1. Schreibe eine funktionale *Spezifikation* für das Problem.
2. Zerlege das Problem in *Unterprobleme* und schreibe für diese Unterprobleme *Spezifikationen*.
3. Beschreibe, wie sich die Lösungen der Unterprobleme zu einer *Gesamtlösung kombinieren* lassen. Beweise die *Korrektheit* der Gesamtlösung unter der Voraussetzung, daß die Teillösungen korrekt sind. Schätze den *Aufwand* des A. ab.
4. Entwerfe nach dem gleichen Schema *Unteralgorithmen* für die nichtelementaren Unterprobleme.

Art Umkehrung der Top–Down–Methode ist die **Bottom–Up–Methode**:

Start mit Teillösungen; Gesamtlösung aus diesen.

Klar: In Praxis nie reines Verfahren bis zum Ende.

■ **Beispiel 2.6** – Fortsetzung: Suche nach einem Objekt

Gesucht ist die Position P(a, A) eines Elementes a in einer Folge A. Es gelte P(a, A) = 0 für $a \notin A$.

1. Deklaration:

Datentypen: M – eine Menge;
A(M) – Menge aller endl. Folgen über M
schreiben Folge A als $A = (a_1, a_2, ..., a_n)$

Funktionen: $P : M \times A(M) \rightarrow \mathbf{N}$

Variablen: $p \in \mathbf{N}$

2. Eingabe: $A = (a_1, a_2, ..., a_n) \in A(M)$
$a \in M$
es sei $n \geq 1$

3. Vorbedingung: $a_i \neq a_j$ für $i \neq j$, $i, j = 1 \ldots n$

4. Verfahren**:** ***Grobentwurf***

1. *Wähle eine erste Suchposition* p.
2. Falls $a = a_p$ gilt, brich Suche ab.
   Andernfalls, wenn noch *eine neue Suchposition existiert, wähle eine neue Suchposition* p
   und starte 2. erneut. Wenn keine neue Suchposition mehr existiert, setze p gleich 0 und brich ab.

5. Ausgabe: p := P(a, A)

6. Nachbedingung:

$$(a = a_p \wedge 1 \leq p \leq n) \vee ((\forall j \in \{1,\ldots,n\}: a \neq a_j) \wedge p = 0)$$

Bem.:
Grobentwurf des Verfahrens enthält drei Unterprobleme:

1. *Wähle eine erste Suchposition.*
2. *Existiert eine neue Suchposition?*
3. *Wähle eine neue Suchposition.*

➔ Entwurfsentscheid. im folg. Verfeinerungsschritt

⇨ **Regel der minimalen Festlegung:**
Es sollte das Unterproblem zuerst bearbeitet werden, dessen Lösung von den übrigen am wenigsten abhängt und so deren Lösung am geringsten festlegt.

Hier: Unterproblem 1 „*Wähle eine erste Suchposition*“.

Klar: Beginnen mit Suche an den Enden, also mit p = 1 oder p = n.

Betrachten hier nun die beiden naheliegenden Varianten:

Wähle 1. Position	*∃ neue Position*	*Wähle neue Position*
Setze p auf 1	p < n ?	Ersetze p durch p+1
Setze p auf n	p > 1 ?	Ersetze p durch p–1

Idee: Führen einen sogenannten *Stopper* ein.

➔ Test p < n bzw. p > 1 ersetzt durch p = n+1 bzw. p = 0 nach HALT.

➔ 1.Verfeinerung des Verfahrens:
{Stopper als Element an 0. Position}

1. Setze p gleich n.
2. Wenn a = a_p gilt, beende Suche. Andernfalls ersetze p durch p–1. Falls p = 0, beende Suche, sonst starte 2. erneut.

Weitere Überlegung:

Solange $a_p \neq a$ ist wird Suche fortgesetzt. Dazu den A.-baustein Folge (Sequenz).

➔ Frage: Welche?

➔ Antwort: Für n = 0 wird Suche beendet **bevor** erstmals der Schleifenkörper betreten wird.

➔ Abweisschleife!

⇨ 2. Verfeinerung des Verfahrens (in Pseudocode):

```
a_0 ← a;
p ← n;                    {Wähle eine erste Suchposition}
SOLANGE   a_p ≠ a
FUEHRE AUS
        p ← p–1;          {Wähle neue Suchposition}
schreib(p);
```

➔ Wie Spezifikation ändern? (Deklar., Vorbed., Nachbed.)

Zusammenfassung:

- Die Top–Down–Entwurfsmethode strebt, ausgehend von einem ersten Grobentwurf, durch schrittweises Verfeinern eine Lösung an.
- Die schrittweise Verfeinerung erfolgt nicht im luftleeren Raum.
- Entwickler muß Fähigkeiten des Prozessors kennen.
- Ende der Verfeinerung, wenn jeder Schritt in entsprechender PS ausgedrückt.
- folg. Prinzipien einer guten Programmierung:
    1. Wahl problembezogener Worte für frei wählbare Bezeichner
    2. Einfügen geeigneter Kommentare
    3. Wahl problemangepaßter Datenstrukturen
    4. Strukturierung des Textes, um Steuerfluß sichtbar zu machen