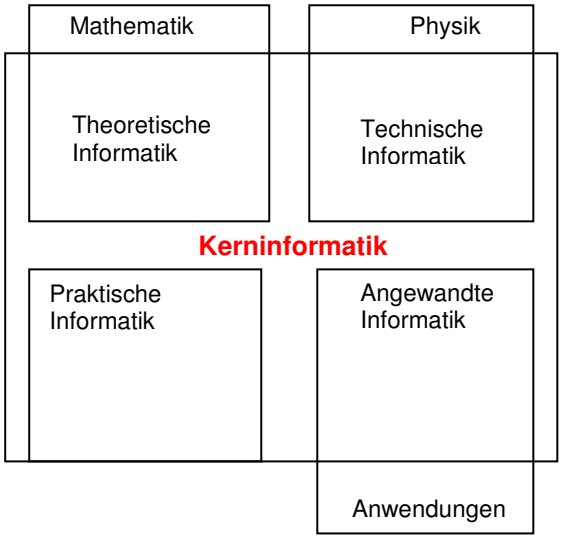
Grundbegriffe Algorithmen

Informatik

- Wissenschaft von der EDV , maschinelle Informationsverarbeitung)
- Konzepte, unabhängig von Rechnertechnologie
- Formalismus, Codierung
- interdisziplinärer Charakter



Kapitel 1. Algorithmen als Teil der „Kerninformatik“

Die Bezeichnung **Algorithmus** geht auf den arabischen Schriftsteller Abu Dshafar Muhammed Ibn Musa al-Khwarizmi zurück, der um 780 in der Stadt Khiva im heutigen Usbekistan geboren wurde (Khiva hieß damals Khwarizm und wurde als Teil des Namens verwendet), lebte und arbeitete später aber vor allem in Bagdad.

Al-Khwarizmi beschrieb u.a. die Erbschaftsverhältnisse, die sich ergaben, wenn ein wohlhabender Mann starb, der bis zu vier Frauen in unterschiedlichem Stand und eine Vielzahl von Kindern hatte. Dazu verwendete er algebraische Methoden und schrieb ein Lehrbuch mit dem Titel "Kitab al jabr w'almuqabalah' (Regeln zur Wiederherstellung und Reduktion), wobei die Übertragung von Gliedern einer Gleichung von der einen zur anderen Seite des Gleichheitszeichens gemeint ist. Der Begriff **Algebra** leitet sich aus dem Titel des Lehrbuchs ab. Aus dem Namen des Schriftstellers wurde Algorism und daraus Algorithmus.

Definitionen Algorithmus, Programm, Prozess, Compiler

Def. Algorithmus: Eine endlich lange Vorschrift, die aus Einzelanweisungen besteht. D.h., ein Algorithmus ist eine genaue, eindeutige und endliche Beschreibung eines endlichen Prozesses (→ systematische Abfolge von Handlungsanweisungen).

Diese Beschreibung erfolgt in einem beliebig wählbaren Formalismus (Notation, Schreibweise), in Termen anderer Algorithmen und, letztlich, in elementaren Algorithmen, z.B. Gebrauchsanweisung, Montageanleitung, Strickmuster, Kochrezept, Funktionsplan einer Kasse usw.

Entscheidend: Der Ausführende muss die Bedeutung der Einzelanweisungen kennen.
Sekundär: Er muss erst mal nicht wissen, was dabei entsteht.

D.h., der Algorithmus muss von einem Prozessor (→ der Ausführende: Mensch oder Automat) ausführbar sein, d.h. verstanden werden. Der Prozessor muss den Formalismus kennen und die elementaren Algorithmen beherrschen (→ Elemente einer Programmiersprache). Der Algorithmus erlaubt es, über den Prozess und sein Ergebnis Aussagen zu machen. Zu gleichen Eingabewerten liefert der Algorithmus stets die gleichen Ausgabewerte. **Trivialalgorithmen:**

Telefonieren:	1. Hörer abnehmen
	2. Geld einwerfen
	3. Wählen
	4. Sprechen
	5. Auflegen **[*Ende*]**
Kochrezept:	1. Man nehme einen Topf Wasser
	2. Man erhitze das Wasser, bis es kocht
	3. Wenn das Wasser kocht, gib Reis hinein **[*usw.*]**
Montageanleitung:	1. Stecke die Tischbeine A-B-C-D in die Platte E
	2. Fixiere A-B-C-D und E durch die Schrauben 1 bis 4 **[*usw.*]**

Der Durchführende kennt die Bedeutung der Einzelanweisungen; sie werden deterministisch, nicht zufällig abgearbeitet. Endliche Vorschrift bedeutet nicht endliche Laufzeit, aber die Beschreibung der Vorschrift muss endlich sein.

Programm		Maschinenprogramm
if (a > b)	Compiler	011011101110110111

Wenn der Computer derjenige ist, der die Einzelanweisungen ausführt, dann gilt:
Def. Programm: Algorithmus, der für den Computer formuliert wurde.
Def. Prozess: Programm in Ausführung.

Die Einzelanweisungen müssen für den Computer verständlich (lesbar) sein. Sie müssen so formuliert sein, dass der Rechner sie ausführen kann. Dazu dient der Compiler („Übersetzer“).

Def. Compiler: → ein Computerprogramm, das ein in einer Quellsprache geschriebenes Programm in ein semantisch äquivalentes Programm einer Zielsprache umwandelt (→ Sequenz von Einzelanweisungen, die der Computer ausführen kann). Üblicherweise handelt es sich dabei um die Übersetzung eines von einem Programmierer in einer Programmiersprache geschriebenen Quelltextes in die Maschinensprache. Das Übersetzen eines Quellprogramms in ein Zielprogramm durch einen Compiler wird auch als Kompilierung bezeichnet.

Kapitel 2: Anweisungenfolge und Anweisungsarten

Im A. muss die systematische **Folge von Einzelanweisungen** fünf Bedingungen erfüllen:

1. **Allgemeingültigkeit:** Die Anweisungen besitzen Gültigkeit für die Lösung einer ganzen Problemklasse, nicht nur für ein Einzelproblem.
2. **Ausführbarkeit:** Die Anweisungen müssen verständlich formuliert sein für den Befehlsempfänger (Mensch oder Maschine) und für diesen ausführbar sein.
3. **Eindeutigkeit:** An jeder Stelle muss der Ablauf der Anweisungen eindeutig sein.
4. **Endlichkeit:** Die Beschreibung der Anweisungsfolge muss in endlichem Text möglich sein.
5. **Terminiertheit:** Nach endlich vielen Schritten liefert die Anweisungsfolge eine Lösung.

Darüber hinaus gibt es im Algorithmus verschiedene **Arten von Anweisungen**:

1. Elementare Anweisungen

es fehlt die Möglichkeit, Entscheidungen zu treffen!
keine strukturierte Sequenz möglich

Beispiele:

teile x durch 2

oder:

erhöhe y um 1

2. Strukturierte Anweisungen (mit Verzweigungen)

enthalten Kontrollstruktur, Bedingung, Teilanweisung
strukturierte Sequenz möglich
Einbeziehung anderer Sequenzen möglich

Trivialalgorithmus

1. Nimm Hörer ab
2. Wähle

    3. **WENN** Freizeichen,
       DANN warte 1 Minute.
       nach 1 Minute Wartezeit lege Hörer auf.

    4. **SOLANGE** kein Freizeichen,
       lege Hörer auf und
       beginne erneut bei #1.

Pseudocode (s.S.9)

```
lies x                          → gib ganze Zahl ein (wird in Variable x abgespeichert)
setze z auf 0                   → setze den Wert im Speicher z auf 0
SOLANGE x ≠ 1 TUE:              → z.B. x = 3
      WENN x gerade
      DANN halbiere x
      SONST verdreifache x und erhöhe x zusätzlich um 1  → aus 3 wird 10
erhöhe z um 1
gib z aus                       → Aufgrund der Anfangseingabe 3 folgt Endausgabe 7
```

Aufgabe 1
Erstelle ein Ablaufprotokoll („Trace“) der o.a. Schleife infolge der Eingabe von Zahl 3. Auf die Eingabe von 3 muss nach vollständigem Ablauf des Algorithmus’ die Ausgabe von Zahl 7 (→ 7 Iterationen: Programmdurchläufe) folgen:

Zeile	x	z
	undef.	undef.
1	**3**	undef.
2	3	0
3	10	0
4	10	1
3	5	1
4	5	2
3	16	2
4	16	3
3	8	3
4	8	4
3	4	4
4	4	5
3	2	5
4	2	6
3	1	6
4	1	7
5	Ausgabe	**7**

Programmablauf in JAVA (nur Kernalgorithmus)

```
{
int x, z;                                        // definiere 2 Variablen
x = IO.readInt("Bitte eine Zahl: ");             // lies einen Wert ein
z = 0;                                           // setze z auf 0
while (x != 1) {                                 // solange x ungleich 1 ist
          if (x % 2 == 0)                        // falls x gerade ist
             x = x / 2;                          // halbiere x
          else                                   // andernfalls
             x = 3*x+1;                          // verdreifache x und add. 1
          z = z+1;                               // erhoehe z um eins
           }
IO.println("Anzahl der Iterationen: " + z);      // gib z aus
}
```

Aufgabe 2
Erstelle das Ablaufprotokoll der Schleife infolge der Eingabe von Zahl 11: Wie viele Durchläufe?

Beispiel: Bestimmung von Primzahlen

Eingabe von x ∈ ℕ	(natürliche Zahl)
Programm: Ist x eine Primzahl?	(kann kein Produkt zweier natürlicher Zahlen, die größer als 1 sind, sein: 2, 3, 5, 7, 11…)
Ausgabe {ja, nein}	

Programm löst Problem nach **EVA-Prinzip:**

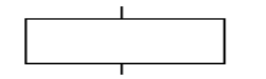
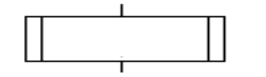
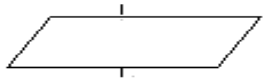
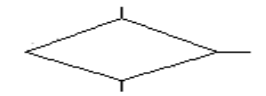
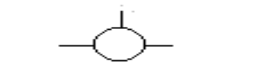
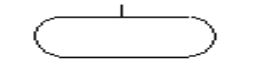
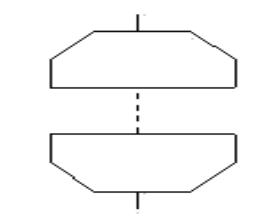
Eingabe	→ Natürliche Zahl x
Verarbeitung (Programm)	→ Ist x z.B. eine Primzahl?
Ausgabe	→ {ja, nein}

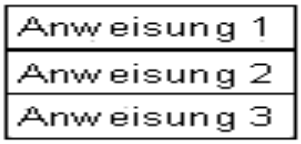
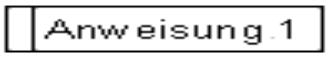
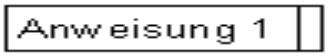
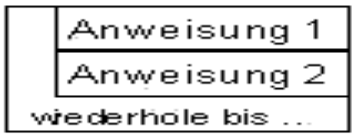
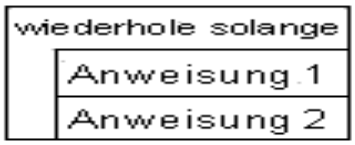
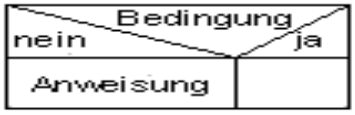
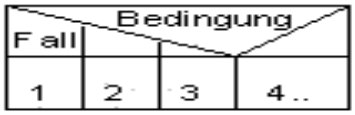
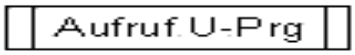
1. Terminierung: Sind wir sicher, dass das Programm zu einem Ende kommen kann?
2. Korrektheit: Tut das Programm auch das, was wir wollen?
3. Komplexität: Wie lange läuft das Programm?

Kapitel 3: Algorithmen und ihre sinnbildliche Darstellung

Mit einem Computer sind nur solche Probleme lösbar, zu denen ein sinnbildlich darstellbarer Algorithmus vorhanden ist. Ein Algorithmus, der schließlich in einer für den Computer verständlichen Sprache formuliert ist, ist ein Programm.

Sinnbild, Programmablaufplan und Struktogramm: Die Programmdokumentation ist eine notwendige Voraussetzung zur Entwicklung und Wartung eines Programmes. Vor allem müssen die Programme grafisch dargestellt werden, weil nur so ein Überblick möglich ist. Speziell dafür verwendet man genormte Sinnbilder nach DIN 66001 (Programmablaufplan) oder auch DIN 66261 (Struktogramme nach Nassi/Shneiderman).

Sinnbild DIN 66001	**Benennung**
	Operation allgemein
	Unterprogrammaufruf
	Ein- bzw. Ausgabe
	Verzweigung
	Übergangsstelle
	Grenzstelle
	Ablauflinien
	Schleifenbegrenzung für zählergesteuerte Wiederholung

Sinnbild DIN 66261	**Benennung**
	Folgeblock für Wertzuweisungen, Rechenoperationen, Bildschirmbefehle
	Eingabeanweisung
	Ausgabeanweisung
	Wiederholung mit Endebedingung
	Wiederholung mit Anfangsbed.
	Verzweigung a) einseitig b) zweiseitig
	Verzweigung mehrfach
	Aufruf Unterprogramm a) Prozedur b) Funktion

Bei umfangreichen Programmen mit vielen Programmverzweigungen erweist sich das **Struktogramm** als vorteilhaft. Es ist übersichtlicher. Das liegt auch daran, dass für Programmablaufpläne keine Symbolik genormt ist, mit der man eine Mehrfachverzweigung skizzieren kann.

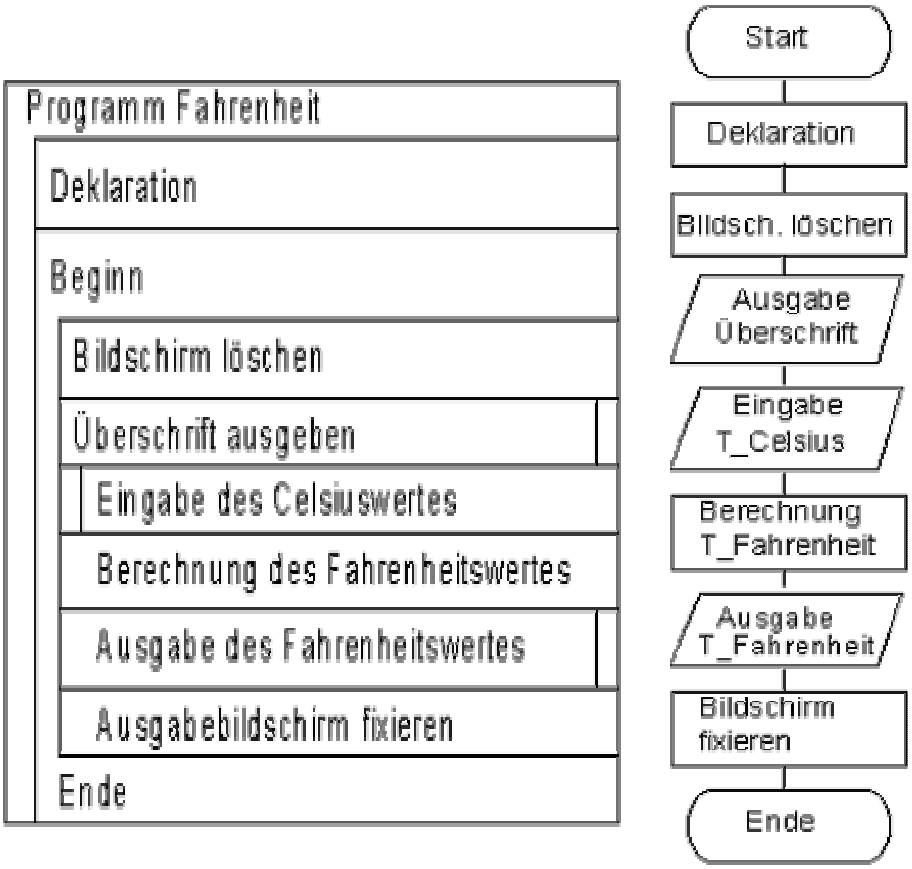
Nicht nur für das Hauptprogramm müssen Ablaufpläne gezeichnet werden, sondern für jedes Unterprogramm zusätzlich. Das heißt, ein Programm mit z.B. 5 Unterprogrammen basiert auf der Grundlage von 6 Ablaufplänen (auf einem fürs Hauptprogramm und fünf für die Unterprogramme).

Beispiel „Fahrenheit“:

In Amerika wird die Temperatur nicht in Grad Celsius, sondern in Grad Fahrenheit gemessen. Es gilt : 0 °C = 32 °F; 100 °C = 212 °F

Aufgabe: Entwirf ein Struktogramm für ein Programm, das nach Eingabe des Temperaturwertes in Grad Celsius den zugehörigen Wert in Grad Fahrenheit auf dem Bildschirm sauber geordnet und lesbar ausgibt.

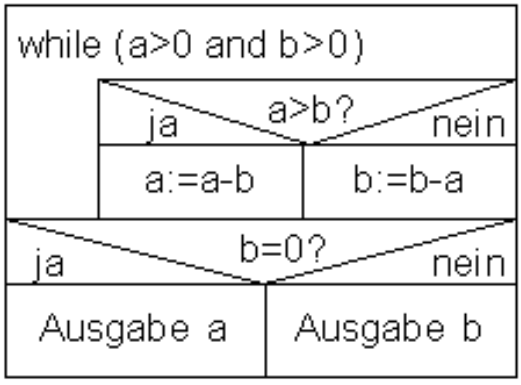
Struktogramm **Programmablaufplan**



a) Einfaches Struktogramm

Beispiel: Euklidischer Algorithmus zur Berechnung des ggT zweier Zahlen.

als **Nassi-Shneiderman-Diagramm** ...



... und in **Pascal**:

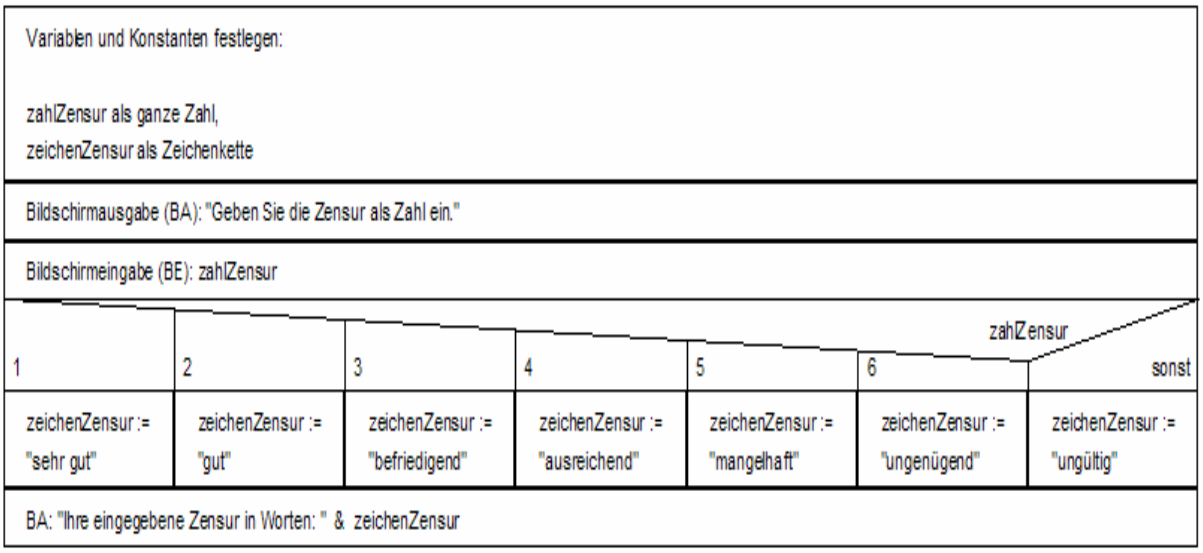
```
PROGRAM GGT(Input,Output);
   VAR a,b: Integer;

BEGIN
   ReadLn(a,b);

WHILE (a > 0) AND (b > 0) DO
   IF a > b THEN
         a := a-b;
   ELSE
          b := b-a;
   IF b = 0 THEN
      WriteLn(a)
ELSE
      WriteLn(b)
END
```

b) Erweitertes Struktogramm

Beispiel: Umbildung einer Notenzahl in den entsprechenden Notentext



... und seine Umsetzung in **Visual Basic**:

```
Option Explicit

Private Sub btnZensur_Click()

  Dim intZensur As Integer, strZensur As String

  intZensur = InputBox("Geben Sie die Zensur als Zahl ein.")

  Select Case intZensur
    Case 1: strZensur = "sehr gut"
    Case 2: strZensur = "gut"
    Case 3: strZensur = "befriedigend"
    Case 4: strZensur = "ausreichend"
    Case 5: strZensur = "mangelhaft"
    Case 6: strZensur = "ungenügend"
    Case Else: strZensur = "ungültig"
  End Select

  MsgBox "Ihre eingegebene Zensur als Text:" & strZensur

End Sub
```

Kapitel 4: Algorithmen und Pseudocode

Der so genannte **Pseudocode** ist eine Mischung aus natürlicher Sprache, mathematischer Notation und einer höheren Programmiersprache. Wie Flussdiagramme und Nassi-Shneidermann-Diagramme ist auch Pseudocode eine Möglichkeit, um Algorithmen verständlicher darzustellen.

Ein Algorithmus wird in Pseudocode einerseits genauer beschrieben als in natürlicher Sprache, anderseits aber noch nicht so detailliert wie durch ein Computerprogramm. Ein Programm in Pseudocode ist geeignet um von Menschen gelesen zu werden, aber nicht geeignet um von einen Computer ausgeführt zu werden. Pseudocode ist also *keine* Programmiersprache.

Für Pseudocode gibt es keine verbindlichen Vorschriften. Jeder kann seine eigene Variante zurechtschneidern. Das Ziel ist, Algorithmen verständlich und klar auszudrücken, ohne auf die Eigenheiten einer Programmiersprache Rücksicht nehmen zu müssen.

Um einen Algorithmus zu verstehen, kann man ihn als Programm untersuchen. Das wird aber erschwert durch die Eigenheiten der Programmiersprache, vor allem ihre Syntax. Zudem haben verschiedene Programmiersprachen unterschiedliche Syntaxen. Jede Formulierung als Programm in einer gewissen Programmiersprache schließt alle Leser aus, die dieser Programmiersprache nicht mächtig sind. Deshalb formuliert man den Algorithmus zwar ähnlich zu einem Programm, aber ohne auf eine bestimmte Programmiersprache einzugehen: in Pseudocode.

Pseudocode wird dann eingesetzt, wenn die Funktionsweise eines Algorithmus erklärt werden soll, und Einzelheiten der Umsetzung in eine Programmiersprache stören würden. Ein typisches Beispiel sind die Felder, die in Pascal von Eins an indiziert werden, in C dagegen von Null an. In Lehrbüchern werden deshalb Algorithmen gelegentlich in Pseudocode wiedergegeben.

Man kann ein Programm durch Pseudocode spezifizieren. Das sollte allerdings eher vermieden werden, denn die Formulierung als Pseudocode ist bereits eine Programmiertätigkeit, die von der Konzentration auf die Anforderungen ablenkt. Auch bei der Entwicklung von Algorithmen und der Umformung von Programmen wird Pseudocode eingesetzt.

Pseudocode hat den Anspruch, intuitiv klar zu sein. Geeignete Metaphern aus der Umgangssprache geben einen Verfahrensschritt prägnant wieder, ohne dass dazu eine Erklärung nötig ist, zum Beispiel "durchlaufe das Feld a mit Index i" oder "vertausche die Inhalte der Variablen x und y". Solche Stilmittel verbessern die Übersicht. Pseudocode kann sich in seinem Stil an eine bestimmte höhere Programmiersprache anlehnen, zum Beispiel an Pascal oder an C.

Im Pascal-Stil werden **Schlüsselworte** wie begin, end, then, do, repeat, until benutzt. Im C-Stil werden stattdessen geschweifte Klammern {,} gesetzt und das Schüsselwort then wird ausgelassen.

Module

program *Programmname* ... end *Programmname*
klasse *Klassenname* { ... }

Fallunterscheidungen

if ... then ... else ... end if/exit
wenn ... dann ... sonst ... wenn_ende
falls ... dann ... falls_nicht ... falls_ende

Schleifen

wiederhole ... solange/bis ... wiederhole_ende
while ... do ...
repeat ... until ...

Kommentare

// kommentar
/* kommentar */

Beispiel Pseudocode „Euklidscher Algorithmus“

➔ ältester bekannter nicht-trivialer Algorithmus (s.o. S. 7)

Euklid (um 300 v.Chr.) berechnete den größten gemeinsamen Teiler (ggT), indem er nach einem gemeinsamen „Maß“ für die Längen zweier Linien suchte.

Dazu zog er wiederholt die kleinere der beiden Längen von der größeren ab. Ist die Differenz von *a* und *b* sehr groß, sind unter Umständen viele Subtraktionsschritte notwendig.

Prozedur: euklid
Parameter: natürliche Zahlen m, n
1. Falls $m > n$, dann m und n miteinander vertauschen.
2. Jetzt gilt $m \leq n$.
3. Solange $m > 0$ wiederhole:
4. Setze $n = n - m$.
5. Falls $m > n$, dann m und n miteinander vertauschen.
6. Jetzt gilt $m \leq n$.
Ergebnis: n.

Man nimmt bei AB, CD abwechselnd immer das Kleinere vom Größeren weg, dann muss (schließlich) eine Zahl übrig bleiben, die die vorangehende misst: