Übung zu
Algorithmen und Datenstrukturen (für ET/IT)

Sommersemester 2014

Jakob Vogel

Computer-Aided Medical Procedures
Technische Universität München

Euklidischer Algorithmus

„Wenn CD aber AB nicht misst, und man nimmt bei AB, CD abwechselnd immer das kleinere vom größeren weg, dann muss (schließlich) eine Zahl übrig bleiben, die die vorangehende misst.“

aus *Euklid: Die Elemente, Buch VII (Clemens Thaer)*

- Vorschrift zur **Berechnung** des *größten gemeinsamen Teilers*
- Keine **Definition** des ggT im mathematischen Sinn
- Verschiedene Algorithmen mit gleichem Ziel ($\rightsquigarrow$ Sortierung!)

Ablauf nach Originalformulierung

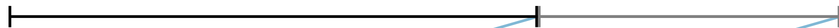
$AB = 11$

$CD = 4$

0 Rechenschritte

Ablauf nach Originalformulierung

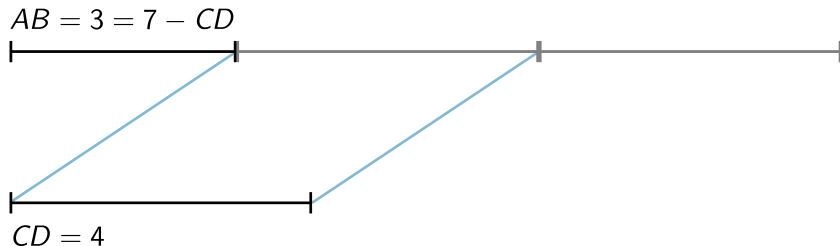
$AB = 7 = 11 - CD$



$CD = 4$

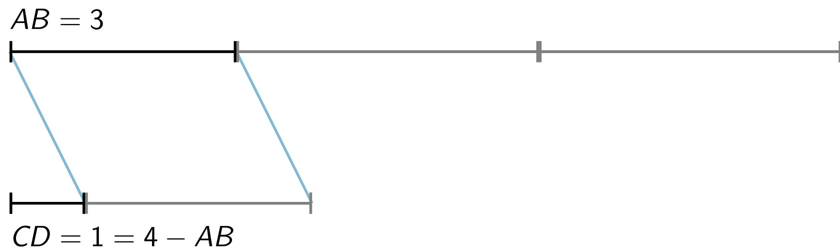
1 Rechenschritt

Ablauf nach Originalformulierung



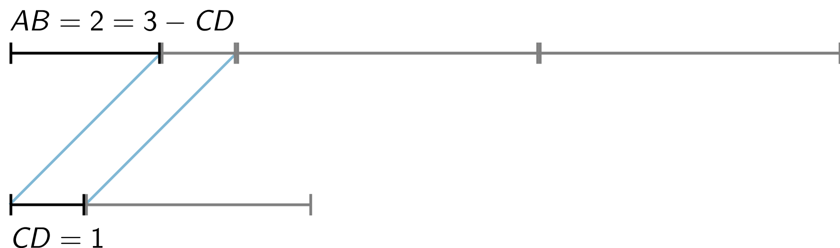
2 Rechenschritte

Ablauf nach Originalformulierung



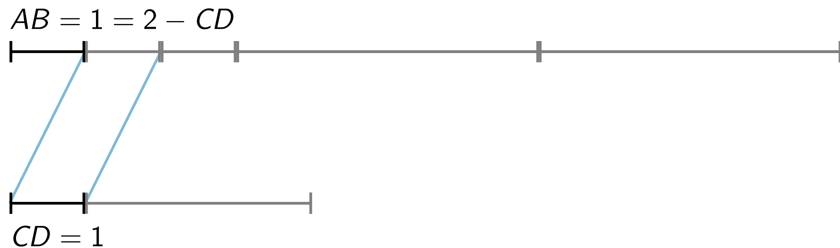
3 Rechenschritte

Ablauf nach Originalformulierung



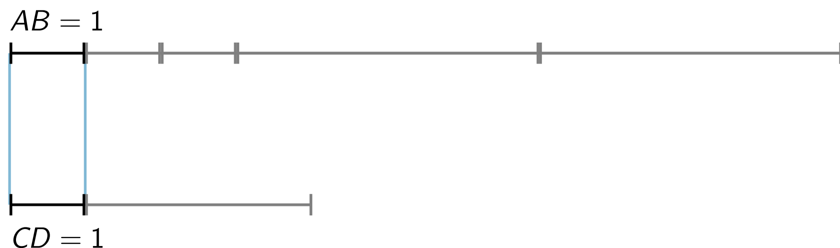
4 Rechenschritte

Ablauf nach Originalformulierung



5 Rechenschritte

Ablauf nach Originalformulierung



$\Rightarrow ggT(11, 4) = 1$

11 ist eine Primzahl!

5 Rechenschritte

Umsetzung auf Rechner

- Programmiersprachen zur formalen Beschreibung (Schlüsselworte, Symbole, Konstrukte)
- Theoretische Unterschiede: Struktur und Abstraktionsgrad
- Technische Unterschiede: Interpretierung und Kompilierung

C++

- Kompilierte Sprache, sehr effizient
- Einfacher und sicherer als C (aber kompatibel!)
- **Industriestandard,** viele Bibliotheken

Entwicklungsumgebungen

- *Microsoft Visual Studio* für Windows
  - Express-Version von `http://microsoft.com/express`
  - Vollversion von `https://maniac.tum.de/`
- *Xcode* für Mac OS X
  - `https://itunes.apple.com/de/app/xcode/id497799835`
- *Qt Creator* für Windows, Mac OS X, GNU/Linux
  - `https://qt-project.org/wiki/Category:Tools::QtCreator`
- *Eclipse* für Windows, Mac OS X, GNU/Linux
  - `http://eclipse.org/downloads/packages/eclipse-ide-cc-developers/keplersr2`
- Walkthroughs auf der Website, Hilfe durch die Tutoren

Grundstruktur

- Befehle als Text in einer *.cpp* Datei
- Kompilierung in ein ausführbares Programm
- Kontrollübergabe vom Betriebssystem an das Programm
  - Eigentlich eine Funktion (siehe unten!)
- Befehle mit ; abschließen!
- Kommentare: `//...` (bis Zeilenende) und `/*...*/`

```
int main(int argc, char** argv)
{
  // Hier stehen dann die eigentlichen Befehle
  // des Programmes
  return 0;
}
```

Code

Text-Ausgabe

- Text-Ausgabe für Feedback
- Standard-Bibliothek: `iostream` für Ein- und Ausgabe
- Text als `"String"`, Zeilenvorschub mit `std::endl`

```
#include <iostream>

int main(int argc, char** argv)
{
  std::cout << "Hallo, Welt!" << std::endl;
  return 0;
}
```

Code

Berechnungen

- Mathematische Operatoren +, −, ∗, /, ...
- Verkettung im Ausgabestrom mit Operator <<
- Weitere Operatoren-Klassen existieren

```
#include <iostream>

int main(int argc, char** argv)
{
  std::cout << "5␣+␣" << 3 << "␣=␣"
    << 5+3 << std::endl;
  return 0;
}
```

Code

Variablen

- Speicherung von Werten mittels Typ-behafteter, eindeutig benannter Variablen
  - `int` für ganze Zahlen (Integer)
  - `double` für Fließkommazahlen (Double Precision Floating Point)
  - `std::string` für Zeichenketten (`#include <string>`)
  - Details hierzu und zu weitere Typen in der Vorlesung!
- Zuweisung über `=`
- Kurze Schreibweise für Updates
  - `i *= 4;` statt `i = i * 4;` (ebenso `+=`, `-=`, `*=`, `/=`, ...)
  - `i++;` statt `i = i + 1;` (ebenso `--`)

```
int i = 23;
int j = 2 * (i-2);          // Klammerung!
std::string s = "Hallo!";
```

Code

Kontrollfluß I

- Logischer Typ (`bool`) mit Werten `true` (wahr, 1) und `false` (unwahr, 0)
- Vergleichs-Operatoren: `==` (gleich), `!=` (ungleich), `<`, `<=`, ...
- Verknüpfungen: `&&` (AND; beide wahr), `||` (OR; mindestens eines von beiden wahr), ...
- Negation: `!a` (NOT)

Kontrollfluß II

- Bedingte Ausführung
  - `if (b) {...}`
  - `if (b) {...} else {...}`
  - `if (b) {...} else if (...) {...} else {...}`
  - Klammern {...} zur Sicherheit immer schreiben
- Schleifen für Wiederholungen
  - `while (b) {...}` (Test vor jeder Iteration)
  - `do {...} while (b);` (Test nach jeder Iteration)

Kontrollfluß III

- Aufzählung von 0 bis 99 (Informatiker zählen ab 0!)

```
int i = 0;        // Initialisierung
while (i < 100) // Schleifenbedingung
{
  std::cout << "i␣=␣" << i << std::endl;
  i++;               // Updateschritt
}
```

- Kürzer mit `for (`⟨Initialisierung⟩`;`⟨Bedingung⟩`;`⟨Update⟩`) {}`

```
for (int i = 0; i < 100; i++)
{
  std::cout << "i␣=␣" << i << std::endl;
}
```

Code

Euklidischer Algorithmus in C++

```
#include <iostream>
int main(int argc, char** argv) {
  int ab = 11, cd = 4;
  if (ab == 0)
    {std::cout << "ggT␣=␣" << cd << std::endl;}
  else {
    while (cd != 0) {
      if (ab > cd)
        {ab = ab - cd;}
      else
        {cd = cd - ab;}
    }
    std::cout << "ggT␣=␣" << ab << std::endl;
  }
  return 0;
}
```

Code

Text-Eingabe

- Daten-Eingabe für Flexibilität
- Lesen aus `std::cin` (Schreiben nach `std::cout`)
- Nutzereingaben **niemals** vertrauen!

```
#include <iostream>

int main(int argc, char** argv)
{
  int i;
  std::cout << "Zahl? ";
  std::cin >> i;
  std::cout << "Eingabe: " << i << std::endl;
  return 0;
}
```

Code

Funktionen I

- Befehlssequenzen zentral hinterlegen statt kopieren! Code strukturieren!
- Definition mittels Name, Rückgabetyp und Liste benannter, Typ-behafteter Parameter (*Signatur*)
- Rücksprung mittels `return`

```
int summe(int a, int b)
{
  return a+b;
}
```

Funktionen II

- Der Einsprungpunkt ist auch nur eine Methode. Die Signatur ist nach Konvention:
  - Der Name ist `main`.
  - Der Rückgabewert ist ein `int`. Ein Wert `!=0` signalisiert dem Betriebssystem einen Fehler.
  - Der erste Parameter, `argc`, enthält die Anzahl der Kommandozeilenparameter. Diese erhält man über den zweiten Parameter, `argv`.

```
int main(int argc, char** argv)
{
  return 0;
}
```

Funktionen III

- Verbesserung der *Implementierung* des Euklidischen Algorithmus:
  - Zahlen-Eingabe durch Nutzer
  - Kapselung in Funktionen
- Kommentierung zur Dokumentation

Code

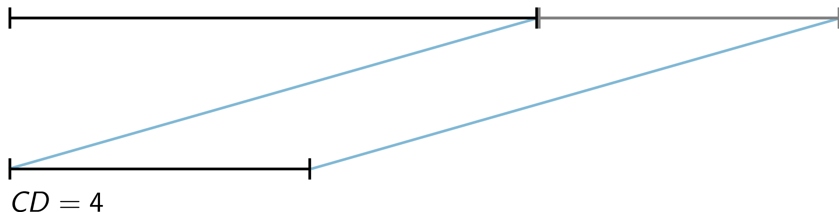
Ablauf nach Originalformulierung

$AB = 11$

$CD = 4$

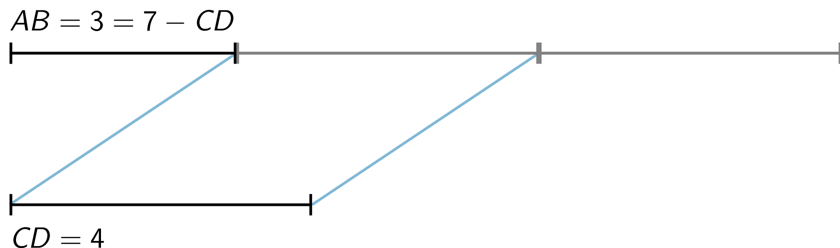
0 Rechenschritte

Ablauf nach Originalformulierung



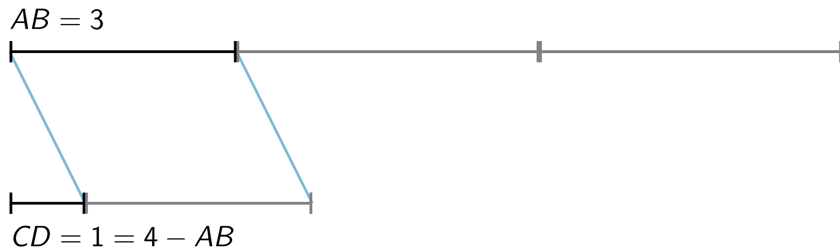
1 Rechenschritt

Ablauf nach Originalformulierung



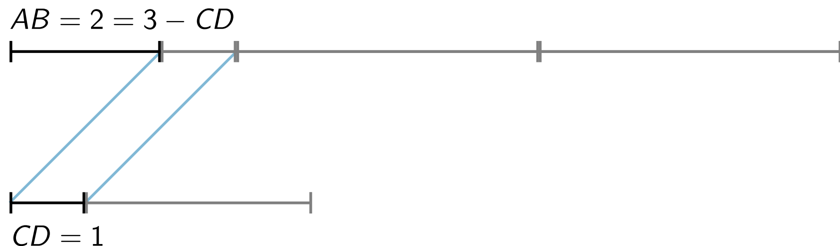
2 Rechenschritte

Ablauf nach Originalformulierung



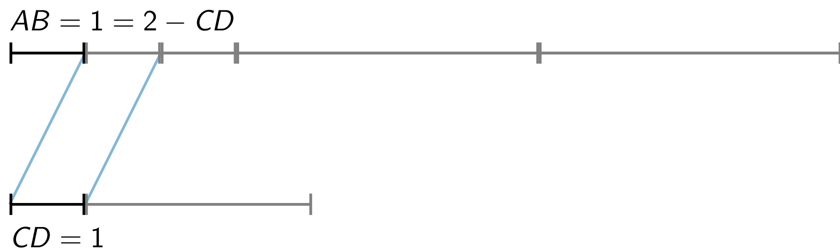
3 Rechenschritte

Ablauf nach Originalformulierung



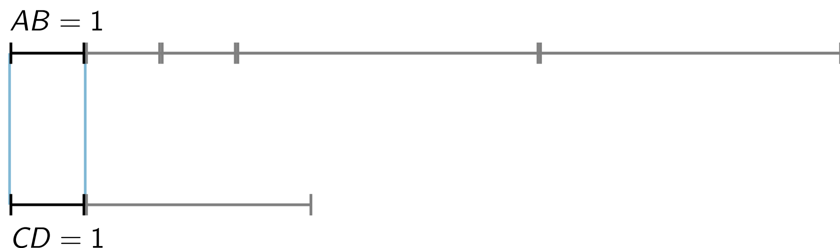
4 Rechenschritte

Ablauf nach Originalformulierung



5 Rechenschritte

Ablauf nach Originalformulierung

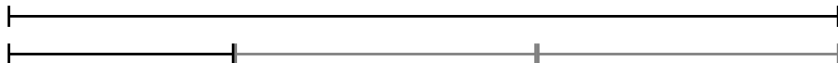


$\Rightarrow ggT(11, 4) = 1$ 11 ist eine Primzahl!

5 Rechenschritte

Beobachtung

$AB = 11 = 2 \cdot CD + 3$



$$
\begin{array}{ccccccccc}
AB & : & CD & = & & 11 & : & 4 & = & 2 \\
 & & & & - & 8 & & & & \\
\hline
 & & & & & 3 & & & &
\end{array}
$$

- Die *Modulo-Berechnung* liefert den Rest nach Teilung.
- Wir schreiben $11 \bmod 4 = 3$ ...
- ... und programmieren `int rest = 11 % 4;`

Ablauf mit Modulo

$AB = 11$

$CD = 4$

0 Rechenschritte

Ablauf mit Modulo

$AB = 3 = 11 \bmod CD$

$CD = 4$

1 Rechenschritt

Ablauf mit Modulo

$AB = 3$

$CD = 1 = 4 \bmod AB$

2 Rechenschritte

Ablauf mit Modulo

$\Rightarrow ggT(11,4) = 1$ Abbruch bei Rest 0!

3 Rechenschritte

Caveat

- Die Zählung von Rechenschritten ist stark vereinfachend!
- Tatsächlich dauert die Berechnung einer Teilung (bzw. einer Modulo-Operation) erheblich länger als eine Differenz!

Code

Bibliotheken I

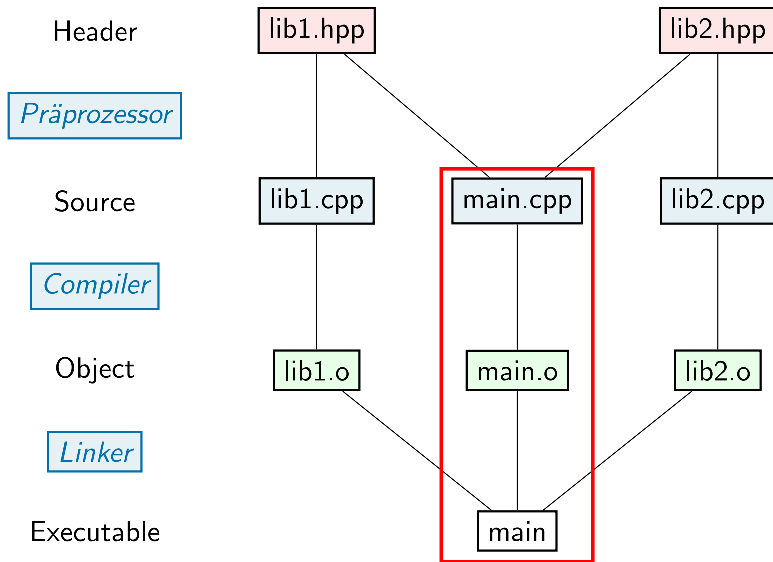
- Bibliotheken sammeln wiederverwendbaren Code
- Die Standard-Bibliothek haben wir bereits benutzt!
  - Weitere Infos etwa unter `http://cppreference.com`
- Weitere Bibliotheken für Dateiverwaltung, Netzwerkzugriff, Grafik, Verschlüsselung, Musteranalyse, ...

Bibliotheken II

- Bibliotheken bestehen aus (kompliliertem) Code und Header (*.h*/*.hpp*).
- Der Header wird mittels `#include` eingebunden.
- Wir lagern `ggT` aus!

☞ Code

Hinter den Kulissen



Kommandozeile unter *NIX

- Kompilieren erzeugt Objekt-Dateien

```
gcc -c main.cpp lib1.cpp lib2.cpp
```

- Linken über

```
gcc -lstdc++ main.o lib1.o lib2.o -o main
```

- Es existieren Abkürzungen!
- Dieser Prozess ist für „Liebhaber“, alle anderen nutzen IDEs!

Ausblick C++

- Für den Anfang nur ein sehr kleiner Ausschnitt aus C++.
- Vieles fehlt:
  - Arrays (Felder)
  - Exceptions (Ausnahmefehler) und Assertions
  - Pointer (Zeiger) und Speichermanagement
  - Dateizugriff
  - ...
- Weitere Konzepte werden in den optionalen Beispielen verwendet werden und dann dort erläutert.
  - Fragen Sie nötigenfalls bei uns oder den Tutoren nach!
- In der Vorlesung sind Programmierkenntnisse nicht zwingend notwendig, im Beruf (wahrscheinlich) schon!

Rekursion

- Funktionen rufen sich selbst wieder auf, nachdem die Parameter geändert wurden.
- Abbruchbedingungen beenden diese Kette von Aufrufen.

- Beispiel: Addition der Zahlen m **bis** n

Iterative Variante

```
int summe_iterativ(int m, int n)
{
  int wert = 0;
  for (int i = m; i <= n; ++i)
    wert += i;
  return wert;
}
```

Rekursive Variante

```
int summe_rekursiv(int m, int n)
{
  if (m > n)
    return 0;
  else
    return m + summe_rekursiv(m+1, n);
}
```

Aufrufkette

- `summe_rekursiv(2, 4)`
- 2 + `summe_rekursiv(3, 4)`
- 2 + 3 + `summe_rekursiv(4, 4)`
- 2 + 3 + 4 + `summe_rekursiv(5, 4)`
- 2 + 3 + 4 + 0
- 2 + 3 + 4
- 2 + 7
- 9

ggT mit Rekursion

- Mittels dreier Eigenschaften läßt sich der ggT rekursiv berechnen:

$$\begin{aligned} \text{ggT}(ab, cd) &= \text{ggT}(cd, ab) && (1) \\ \text{ggT}(ab, cd) &= \text{ggT}(ab, cd \bmod ab) && (2) \\ \text{ggT}(ab, cd) &= \text{ggT}(ab, cd + \lambda \cdot ab) && (3) \end{aligned}$$

Code

Apropos...

- ggT via Primfaktorzerlegung?