TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Der Begriff des Algorithmus

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Der Begriff des Algorithmus I

Zentrale Begriffe der algorithmischen Vorgehens:

- Algorithmus
- Variablen zur Speicherung von Werten
- Ausführungszustand = Speicherzustand + Steuerungszustand
- Zustandsänderung
- Aktion
- Ablauf
- Determinismus, Determiniertheit

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Der Begriff des Algorithmus II

Die prozedurale Modellierung und Programmierung baut auf den klassischen Algorithmusbegriff auf.

Eine Berechnung dabei wird als *zustandsändernder* Ablauf betrachtet.

Damit orientiert sie sich am Berechnungskonzept von Rechnern, das auch auf der Beschreibung von Abläufen basiert, in denen sich in jedem Schritt der Ausführungszustand ändern kann.

Begriffsklärung: Algorithmus

Ein **Algorithmus** ist ein Verfahren zur schrittweisen **Ausführung** von (**Berechnungs-**) **Abläufen**, das sich präzise und endlich beschreiben lässt, so dass:

- die Beschreibung auf wohlverstandenen, ausführbaren ("effektiven") Einzelschritten basiert;
- in jedem Schritt eine oder mehrere Aktionen (ggf. parallel) ausgeführt werden;
- jede Aktion von einem Zustand in einen Nachfolgezustand führt.

Man sagt, die Ausführung eines Algorithmus **terminiert**, wenn sie nach endlich vielen Schritten beendet ist; andernfalls spricht man von einer **nicht-terminierenden** Ausführung.

Beispiel: Algorithmus, der erste I

Verdoppeln nach Adam Riese (1574):


```
Dupliren:
Lehret wie du ein zahl zweyfaltigen solt.
Thu ihm also
 Schreib die zahl vor dich /
 mach ein Linien darunder /
 heb an zu forderst /
 Duplir die erste Figur.
 Kompt ein zahl die du mit einer Figur schreiben magst /
        so setz die unden.
 Wo mit zweyen /
        schreib die erste/ Die ander behalt im sinn.
```


Beispiel: Algorithmus, der erste II

```
Darnach duplir die ander /
 und gib darzu/
 das du behalten hast /
 und schreib abermals die erste Figur /
  wo zwo vorhanden /
 und duplir fort biß zur letzsten /
 die schreibe gantz auß /
 als folgende Exempel außweisen:
        ...
```


Beschreibung eines Algorithmus

Algorithmen lassen sich mit unterschiedlichen Sprachmitteln beschreiben:

- umgangssprachlich
- mit mathematischer Sprache
- in graphischer Notation
- mit programmiersprachlichen Mitteln

Ein Algorithmus ist unabhängig von der verwendeten Beschreibungstechnik bzw. Sprache.

Algorithmus in Umgangssprache

Berechnung des größten gemeinsamen Teilers:

Seien m, n, v Variablen für Integer-Werte.

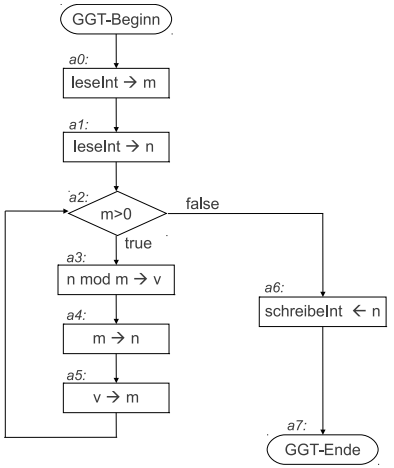
Lese die Werte $w1$ und $w2$ ein, für die der ggT berechnet werden soll und weise $w1$ an m und $w2$ an n zu.

Solange der Wert von m größer als 0 ist, tue Folgendes und prüfe danach wieder die Bedingung:

- Berechne $n \bmod m$ und weise das Ergebnis an v zu;
- Weise den Wert von m an n zu;
- Weise den Wert von v an m zu;

Gebe den Wert aus, den n enthält.

Algorithmus als Flussdiagramm



Algorithmus als Java-Programm

```
public class GGT { // Berechnet ggT fuer 2 gelesene Werte
 public static void main( String[] args ){
   int m, n, v;

   m = Integer.parseInt(args[0]);
   n = Integer.parseInt(args[1]);

   while( m>0 ) {
     v = n % m;
     n = m;
     m = v;
   }
   System.out.println( "ggT:" + n );
 }
```


Begriffsklärung: Speichervariablen I

Um die Zustände zwischen den Schritten präziser fassen zu können, führt man Variablen ein, die Werte speichern können.

Variablen stellen wir graphisch durch Rechtecke dar:

v: `true` v *enthält/speichert* den Wert `true`

x: 7 x *enthält/speichert* den Wert 7

Begriffsklärung: Speichervariablen II

Eine **Speichervariable** (oder einfach nur Variable) ist ein Speicher/Behälter für Werte. Charakteristische Operationen auf einer Variablen v:

- **Zuweisen** eines Werts w an v;
- **Lesen** des Wertes, den v enthält/speichert/hat.

Der Zustand einer Variablen v ist undefiniert, wenn ihr noch kein Wert zugewiesen wurde; andernfalls ist der Zustand von v durch den gespeicherten Wert charakterisiert.

Begriffsklärung: Zustände

Jeder Schritt bei der Ausführung eines Algorithmus führt von einem **Ausführungszustand** zum **Nachfolgezustand**.
Ein Ausführungszustand ist gekennzeichnet durch

- den **Speicherzustand** (im Wesentlichen der Zustand der Variablen);
- den **Steuerungszustand** (vereinfacht gesagt, die Stelle im Programm, an der die Ausführung angekommen ist).

Ein Ausführungsschritt führt zu einer Zustandsänderung, also einer Veränderung von Speicher- und/oder Steuerungszustand.

Begriffsklärung: Aktion

In einem Ausführungsschritt wird üblicherweise eine **Aktion** ausgeführt.
Aktionen sind insbesondere:

- Zuweisungen an Variablen
- Prüfen von Bedingungen
- [Kommunikation mit der Umgebung (Ein- und Ausgabe)]

Die Aktion bestimmt nachfolgende Steuerungszustände bzw. die Terminierung des Algorithmus.

Begriffsklärung: Ablauf

Der **Ablauf** eines Algorithmus zu gegebenen Eingaben wird charakterisiert durch

- die Sequenz der Ausführungszustände
- die Sequenz der ausgeführten Aktionen

Begriffsklärung: deterministisch

Ein Algorithmus heißt **deterministisch**, wenn für alle Eingabedaten der Ablauf des Algorithmus eindeutig bestimmt ist.

Andernfalls heißt er **nicht-deterministisch**.

Beispiel: nicht-deterministischer Algorithmus I

Aufgabe:
Erkenne, ob eine Eingabezeichenreihe über Kleinbuchstaben eines der Worte "otto", "toto", "total" enthält.

Eingabe: Eingabezeichenreihe über Kleinbuchstaben

Ausgabe: ja/nein

Nicht-deterministischer Algorithmus:
Seien

- z die Eingabe und
- $prefixlen = \{ 0, \ldots, \text{länge}(z) - 4 \}$

Beispiel: nicht-deterministischer Algorithmus II

Solange $prefixlen \neq \varnothing$, tue folgendes:

- Wähle ein x aus $prefixlen$ aus;
- $prefixlen = prefixlen \backslash \{x\}$;
- $w =$ "z ohne die ersten x Buchstaben";
- Prüfe, ob w mit "otto", "toto" , "total" beginnt ;
- Falls ja, terminiert der Algorithmus mit "ja";
  andernfalls fahre mit dem nächsten Durchlauf der Schleife fort.

Terminiere mit "nein".

Begriffsklärung: determiniert

Ein Algorithmus heißt **determiniert**, wenn er bei gleichen zulässigen Eingabewerten stets das gleiche Ergebnis liefert.

Andernfalls heißt er **nicht-determiniert**.

Beispiele: Determiniertheit

- Jeder Algorithmus, der eine math. Funktion berechnet, ist determiniert.
- Der nicht-deterministische Algorithmus des obigen Beispiels ist determiniert.

Anweisungen in Java

Begriffsklärung: Anweisung

Anweisungen (engl. *statements*) sind programmiersprachliche Beschreibungsmittel.

- **Einfache** Anweisungen beschreiben Aktionen.
- **Zusammengesetzte** Anweisungen beschreiben, wie mehrere Aktionen auszuführen sind.

Wiederholung: Ausdruck

Ausdrücke sind das Sprachmittel zur Beschreibung von Werten.

Ein **Ausdruck** (engl. **expression**) in Java ist (u.a.)

- ein Literal,
- ein Bezeichner (Variable, Name),
- die Anwendung einer Operation auf einen oder mehrere Ausdrücke,
- oder aufgebaut aus Sprachmitteln, die erst später behandelt werden.

Unterschied: Anweisung / Ausdruck

- Die Auswertung von Ausdrücken liefert ein Ergebnis.
- Die Ausführung von Anweisungen verändert den Zustand. Im Allgemeinen liefern sie kein Ergebnis.

Beispiele für Anweisungen in Java

- Variablendeklaration

  `<Typbezeichner>  <Variablenbezeichner> ;`

- Zuweisungen[1]

  `<Variable>  =  <Ausdruck> ;`

[1] In Java ist eine Zuweisung syntaktisch ein Ausdruck, liefert also einen Wert und zwar das Ergebnis der Auswertung von der rechten Seite der Zuweisung.

Begriffsklärung: L-/R-Wert

In einer Zuweisung (und anderen Sprachkonstrukten) kann eine Variable `v` mit zwei Bedeutungen vorkommen:

$$a = 7 + a;$$

L-Wert ↗ ↖ R-Wert

1. Das Vorkommen links vom Zuweisungszeichen meint die Variable. Man spricht vom ***L*-Wert** (engl. *l*-value) des Ausdrucks v .
2. Das Vorkommen rechts vom Zuweisungszeichen meint den in v gespeicherten Wert. Man spricht vom ***R*-Wert** (engl. *r*-value) des Ausdrucks v.

Die Unterscheidung wird wichtiger, wenn auch links des Zuweisungszeichens komplexere Ausdrücke stehen.

Anweisungsblöcke

Ein **Anweisungsblock (engl. block statement)** ist eine Liste bestehend aus Deklarationen und Anweisungen.

Syntax in Java:

```
{  <DeklAnweisListe>  }
<DeklAnweisListe> → ε
                  |  <Deklaration> <DeklAnweisListe>
                  |  <Anweisung>   <DeklAnweisListe>
```

Semantik:

Stelle den Speicherplatz für die Variablen bereit und führe die Anweisungen der Reihe nach aus.

Beispiel

Anweisungsblock:

```
{ int i; i = 7; int a; a = 27 % i; }
```

Üblicherweise schreibt man Deklarationen und Anweisungen untereinander, so dass ein Textblock entsteht.

```
{
  int i;
  int x;
  i = 34;
  x = i * 5;
}
```


Kontrollfluss: Verzweigungen und Schleifen

Verzweigungsanweisungen

Verzweigungsanweisungen (engl. *branch statements / conditionals*) stellen Bedingungen an die Ausführung einer Anweisung oder wählen einen von mehreren Zweigen zur Ausführung aus.

Wir betrachten hier zunächst zwei Arten von Verzweigungsanweisungen:

- Bedingte Anweisung
- Fallunterscheidung

Bedingte Anweisung

Syntax in Java: `if ( <boolscherAusdruck> ) <Anweisung>`

Semantik:
Werte den boolschen Ausdruck aus. Ist das Ergebnis `true`, führe die `Anweisung` aus.

Beispiel:
Wir gehen in diesen Beispielen davon aus, dass die Variablen `x,y,z` bereits deklariert und initialisiert worden sind.

```
// Absolutwert
if (x < 0) { x = -x; }

// Testet, ob Divisor null ist
if (x != 0) { z = y/x; }

// Sortieren von x und y: x soll kleiner als y sein
if (x > y) {
  int t = x;
  x = y;
  y = t;
}
```

Fallunterscheidung

Syntax in Java:

```
if ( <boolscherAusdruck> )   <Anweisung1>
else                         <Anweisung2>
```

Semantik:
Werte den boolschen Ausdruck aus. Ist das Ergebnis `true`, führe `Anweisung1`, andernfalls `Anweisung2` aus.

Beispiel: Münzwurf

```
if (Math.random() < 0.5) {
  System.out.println("Kopf");
} else {
  System.out.println("Zahl");
}
```

Math.random() liefert eine Zufallszahl aus dem Bereich [0.0, 1.0)

Beispiel: Maximumsberechnung I

Beispiel: Maximum zweier Zahlen

```
// Maximum von x und y
int x, y;
int max;

x = ...;  // Initialisieren von x
y = ...;  // Initialisieren von y

if ( x > y ) {
  max = x;
} else {
  max = y;
}
```

Frage: Wie ermittelt man das Maximum dreier Zahlen (x,y,z)?

Beispiel: Maximum dreier Zahlen

Eine mögliche Implementierung:

```
int x, y;
int max;

x = ...;  // Initialisieren von x
y = ...;  // Initialisieren von y

if ( x > y ) {
  if ( x > z ) {
    max = x;
  } else {
    max = z;
  }
} else {
  if ( y > z ) {
    max = y;
  } else {
    max = z;
  }
}
```


Schleifenanweisungen

Schleifenanweisungen (engl. *loop statements*) steuern die iterative, d.h. wiederholte Ausführung von Anweisungen/Anweisungsblöcken.

Wir betrachten hier die folgenden Schleifenanweisungen:

- while-Anweisung
- do-Anweisung
- Zählanweisung (for-Anweisung) (⇒ nächste Vorlesung)

while-Anweisung

Syntax in Java:

```
while( <boolscherAusdruck> ) <Anweisung>
```

Semantik:
Werte den boolschen Ausdruck aus, die sogenannte **Schleifenbedingung**.
Ist die Bedingung erfüllt, führe die Anweisung aus, den sogenannten **Schleifenrumpf**, und wiederhole den Vorgang.
Andernfalls beende die Ausführung der Schleifenanweisung.

Beispiel: ggt

Aus der Prozedur zur Berechnung des ggT:

```
while( m > 0 )
{
   v = n % m;
   n = m;
   m = v;
}
```

```
while( m > 0 ) {
   v = n % m;
   n = m;
   m = v;
}
```

Bemerkung:

Der Schleifenrumpf ist in den meisten Fällen ein Anweisungsblock.
Syntaktisch korrekt ist aber auch z.B.:

```
while( true )   System.out.println(i);
```


Beispiel: Berechnung von Zweierpotenzen

```
/*
* Berechnet die Tabelle aller Zweierpotenzen,
* die kleiner als 2^n sind.
* Der Wert von n wird dabei als Befehlszeilenargument
* uebergeben.
*/

public class Zweierpotenz {
  public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    int v = 1;
    int i = 0;

    while (i < n) {
      System.out.println(i + " " + v);
      v = 2 * v;
      i = i + 1;
    }
  }
}
```


Terminierung

- Durch Programmierfehler ergeben sich bisweilen Situationen, in denen eine Schleife unendlich oft ausgeführt wird ("Endlosschleife").
- Ein Programmierer muss sich daher immer die folgende Frage stellen:

  Wird die Ausführung der Schleife irgendwann abgebrochen?

  - Welche Werte müssen die Variablen in der Schleifenbedingung annehmen, damit diese nicht mehr erfüllt ist?
  - Wie verändern sich die Werte dieser Variablen in jedem Schleifendurchlauf?
  - Wird die Schleifenbedingung daher irgendwann nicht mehr erfüllt?
- Als Hilfestellung kann man sich überlegen, welche Werte die Variablen annehmen, nachdem die Schleife das erste Mal, das zweite, etc., das letzte Mal durchlaufen wird.

Beispiel: Zweierpotenz

- Die Schleifenbedingung ist nicht mehr erfüllt, wenn die Variable `i` größer oder gleich der Eingabe `n` ist.
- In jedem Schleifendurchlauf wird der Wert der Variablen `i` um 1 erhöht, der Wert von `n` verändert sich nicht.
- Fallunterscheidung:
  - *1. Fall* `n` ≤ 0: Da Variable `i` mit 0 initialisiert wird, wird die Schleifenbedingung nie erfüllt, und der Schleifenrumpf nicht ausgeführt.
  - *2. Fall* `n` > 0: Da `i` mit 0 initialisiert wird und in jedem Schleifendurchlauf um 1 erhöht wird, gilt nach dem k-ten Schleifendurchlauf, dass `i` den Wert k hat. Nach dem n-ten Schleifendurchlauf hat `i` also den Wert n, und die Schleifenbedingung ist dann nicht mehr erfüllt.
- Diese beiden Fälle decken alle möglichen Eingabewerte für `n` ab; daher terminiert die Ausführung für alle Eingaben.

do-Anweisung

Syntax in Java:

```
do <Anweisung> while( <boolscherAusdruck> ) ;
```

Semantik:

Führe die Anweisung aus.
Werte danach den booleschen Ausdruck aus.
Ist die Bedingung erfüllt ist, wiederhole den Vorgang.
Andernfalls beende die Ausführung der Schleifenanweisung.

Bemerkung

Die do-Anweisung ist immer dann sinnvoll, wenn man einen Vorgang mindestens einmal ausführen muss.

In solchen Situationen spart man sich gegenüber der Verwendung der while-Schleife die anfänglich unnötige Auswertung der Bedingung.

Sprung- und Auswahlanweisungen

Sprunganweisungen

Sprunganweisungen (engl. *jump statements*) legen eine Fortsetzungsstelle der Ausführung fest, die möglicherweise weit von der aktuellen Anweisung entfernt liegt.

Wir betrachten hier nur Sprünge, die der Programmstruktur folgen.

Abbruchanweisung

Syntax in Java: `break;`

Semantik:
Die Ausführung wird mit der Anweisung fortgesetzt, die nach der umfassenden zusammengesetzen Anweisung kommt.

Typische Einsatzgebiete:

1. In Auswahlanweisung: siehe unten.
2. In Schleifenanweisungen:

```
while( true ) {
    ...
    if( <Abbruchbedingung> ) break;
    ...
}
```


Auswahlanweisung

Auswahlanweisungen erlauben es, in Abhängigkeit vom Wert eines Ausdrucks direkt in einen von endlich vielen Fällen zu verzweigen.

In Java gibt es dafür die `switch`-Anweisung.

Beispiel:

```
String eingabe = args[0];
switch( eingabe ) {
  case "a":  System.out.println("A wie Apfel");       break;
  case "b":  System.out.println("B wie Banane");      break;
  case "c":  System.out.println("C wie Clementine"); break;
  default:   System.out.println("Falsches Eingabezeichen");
  }
```


Übersicht: Anweisungen

- Einfache Anweisungen (Zuweisung, Prozeduraufruf)
- Anweisungsblöcke
- Schleifenanweisungen (while-, do-, for-Anweisungen)
- Verzweigungsanweisungen:
  - bedingte Anweisung
  - Fallunterscheidung
  - Auswahlanweisung
- Sprunganweisungen (Abbruchanweisung)

Weitere Arten von Anweisungen werden im Verlauf der Vorlesung noch behandeln.