

Script generated by TTT

Title: Einf_HF (16.06.2014)

Date: Mon Jun 16 14:17:13 CEST 2014

Duration: 86:36 min

Pages: 32

Systematischer Aufbau von Programmen durch Zerlegung in eigenständige Einheiten.

- [Allgemeines](#)
- [Beispiel Zerlegung](#)
- [Strukturierung von Algorithmen](#)
- [Module](#)
- [Prozedurales / Objektorientiertes Programmieren](#)

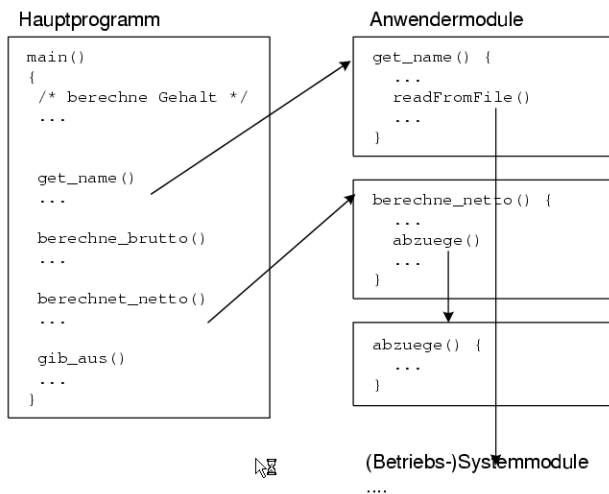
Generated by Targeteam



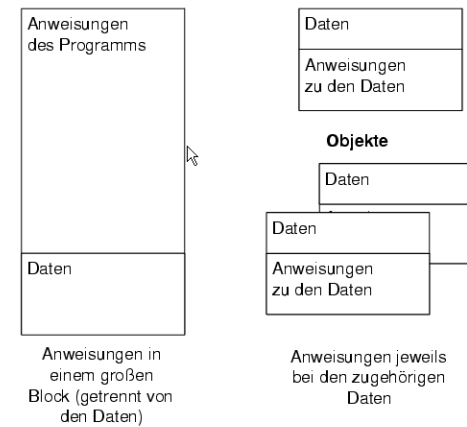
Hauptprogramm bestehend aus Teilmodulen



Prozedurales / Objektorientiertes Programmieren



Prozedurales Programmieren Objektorientiertes Programmieren



Prozedurales Programmieren

Sequenz von Anweisungen, ausführen, warten. Basiselemente: Sequenz, Auswahl (if-then-else), Iteration. Trennung von Daten und Anweisungen.

Objektorientiertes Programmieren

Objekte statt Anweisungen. Objekt hat Attribute, Methoden und Ereignisbehandlung. Zusammenfassung von Daten und Anweisungen



Systematischer Aufbau von Programmen durch Zerlegung in eigenständige Einheiten.

Allgemeines

Beispiel Zerlegung

Strukturierung von Algorithmen

Module

Prozedurales / Objektorientiertes Programmieren

Spezielle Form der Modularisierung. Zu definierendes Modul wird in seiner Definition selbst benutzt.

"natürlichere" Darstellung bei bestimmten Algorithmen und Datenstrukturen.

Beispiel Summe

Definiertes Ende einer rekursiven Schachtelung.

```
if (n > 0)
    summe = summe(n-1) + n;
else summe = 0; /* definiertes Ende, wenn n <= 0 */
```

Generated by Targeteam

Beispiel 'Fibonacci-Zahlen'

Beispiel 'Größter gemeinsamer Teiler'

Beispiel 'Türme von Hanoi'

Generated by Targeteam



Spezielle Form der Modularisierung. Zu definierendes Modul wird in seiner Definition selbst benutzt.

"natürlichere" Darstellung bei bestimmten Algorithmen und Datenstrukturen.

Beispiel Summe

Definiertes Ende einer rekursiven Schachtelung.

```
if (n > 0)
    summe = summe(n-1) + n;
else summe = 0; /* definiertes Ende, wenn n <= 0 */
```

Beispiel 'Fibonacci-Zahlen'

Beispiel 'Größter gemeinsamer Teiler'

Beispiel 'Türme von Hanoi'

Generated by Targeteam

Übliche mathematische Problemstellung rekursiv formuliert, direkt durch rekursiven Algorithmus lösbar.

Definition

$fib(n) = fib(n-1) + fib(n-2)$ für $n > 2$

$fib(1) = fib(2) = 1$

Wertebereich: $n \geq 1$, mit n natürliche Zahl

Algorithmus

```
Modul fib(n)
Falls n kleiner 3
    dann Ergebnis = 1
sonst Ergebnis = fib(n-1) + fib(n-2)
```

Implementierung in Java

```
static int fib(int n) {
    if (n < 3) return 1;
    else return fib(n-1) + fib(n-2);
}
```

Alternative Implementierung

```
static int fib(int n) {
    return (n < 3) ? 1 : fib(n-1) + fib(n-2);
}
```



Windows Internet Explorer

C:\www\inf-ss14\whiteboard\inf_course6.7.2.html

Beispiel 'Fibonacci-Zahlen'

Übliche mathematische Problemstellung rekursiv formuliert, direkt durch rekursiven Algorithmus lösbar.

Definition

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2) \text{ für } n > 2$$

$$\text{fib}(1) = \text{fib}(2) = 1$$

Wertebereich: $n \geq 1$, mit n natürliche Zahl

Algorithmus

```

Modul fib(n)
  Falls n kleiner 3
    dann Ergebnis = 1
  sonst Ergebnis = fib(n-1) + fib(n-2)
  
```

Implementierung in Java

```

static int fib(int n) {
  if (n < 3) return 1;
  else return fib(n-1) + fib(n-2);
}
  
```

Excel - uebung6-2.xls

Berechnung von Fibonacci Zahlen

fib(1) = fib(2) = 1

Zahl n: 10, Ergebnis: 55

Rekursionsstufen:

0	fib(10)
1	fib(9)
2	fib(8)
3	fib(7)
4	fib(6)
5	fib(5)
6	fib(4)
7	fib(3)
8	fib(2)
8	fib(2) = 1
8	fib(1)
8	fib(1) = 1
7	fib(3) = 2
7	fib(2)
7	fib(2) = 1
6	fib(4) = 3
6	fib(3)
7	fib(2)
7	fib(2) = 1
7	fib(1)
7	fib(1) = 1
6	fib(3) = 2
5	fib(5) = 5
5	fib(4)
6	fib(3)

Excel - uebung6-2.xls

Berechnung von Fibonacci Zahlen

fib(1) = fib(2) = 1

Zahl n: 10, Ergebnis: 55

Rekursionsstufen:

0	fib(10)
1	fib(9)
2	fib(8)
3	fib(7)
4	fib(6)
5	fib(5)
6	fib(4)
7	fib(3)
8	fib(2)
8	fib(2) = 1
8	fib(1)
8	fib(1) = 1
7	fib(3) = 2
7	fib(2)
7	fib(2) = 1
6	fib(4) = 3
6	fib(3)
7	fib(2)
7	fib(2) = 1
7	fib(1)
7	fib(1) = 1
6	fib(3) = 2
5	fib(5) = 5
5	fib(4)
6	fib(3)

Excel - uebung6-2.xls

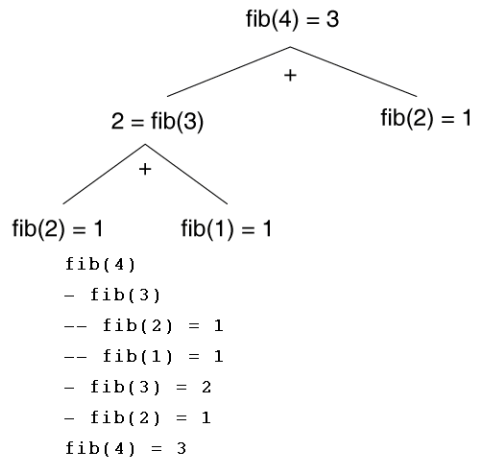
Berechnung von Fibonacci Zahlen

fib(1) = fib(2) = 1

Zahl n: 12, Ergebnis: 144

Rekursionsstufen:

0	fib(12)
1	fib(11)
2	fib(10)
3	fib(9)
4	fib(8)
5	fib(7)
6	fib(6)
7	fib(5)
8	fib(4)
9	fib(3)
10	fib(2)
10	fib(2) = 1
10	fib(1)
10	fib(1) = 1
9	fib(3) = 2
9	fib(2)
9	fib(2) = 1
8	fib(4) = 3
8	fib(3)
9	fib(2)
9	fib(2) = 1
9	fib(1)
9	fib(1) = 1
8	fib(3) = 2
7	fib(5) = 5
7	fib(4)
8	fib(3)
9	fib(2)



Generated by Targeteam

Spezielle Form der Modularisierung. Zu definierendes Modul wird in seiner Definition selbst benutzt.
"natürlichere" Darstellung bei bestimmten Algorithmen und Datenstrukturen.

Beispiel Summe

Definiertes Ende einer rekursiven Schachtelung.

```

if ( n > 0 )
  summe = summe(n-1) + n;
else summe = 0; /* definiertes Ende, wenn n <= 0 */
  
```

Beispiel 'Fibonacci-Zahlen'

Beispiel 'Größter gemeinsamer Teiler'

Beispiel 'Türme von Hanoi'

Generated by Targeteam



Aufruf solveTvH (3, b, a, c) führt zu den nachfolgenden Rekursionen:

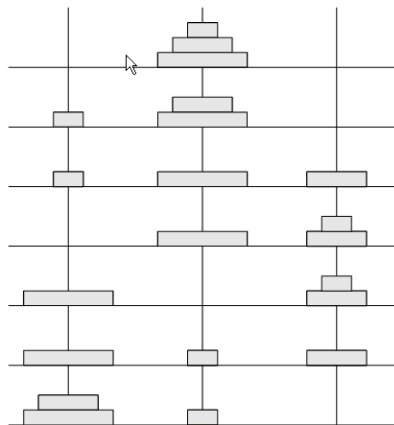
3bac (2bca (1bac (b → a); b → c; 1acb (a → c));

b → a;

2cab (1cba (c → b); c → a; 1bac (b → a)))

3bac entspricht dem Aufruf solveTvH (3, b, a, c)

b → a entspricht: bewege Scheibe von b nach a



TUM MMP WS 02

```

1 function solveTvH(anzahl, start, zwischen, ziel) {
2   if (anzahl == 1) {
3     Setze Scheibe 1 von start nach ziel;
4   } else {
5     solveTvH((anzahl-1), start, ziel, zwischen);
6     Setze Scheibe anzahl von start nach ziel;
7     solveTvH((anzahl-1), zwischen, start, ziel);
8   }
9 }
  
```

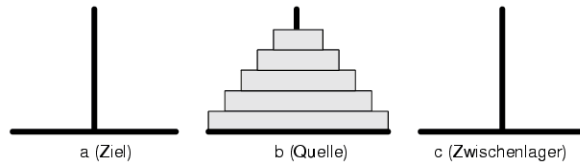
Stack

Scheiben: **3**

Generated by Targeteam



Gegeben: drei senkrechte Stäbe, n Scheiben verschiedener Größe. Aufgabe: Turm von Scheiben von Stab auf anderen übertragen, jeweils nur eine Scheibe bewegen, nie größere Scheibe auf kleinerer.



Rekursive Lösung für n Scheiben

Aufrufschachtelung für n=3

Zahl der Schritte: $2^n - 1$; n ist die Anzahl der zu bewegenden Scheiben.

Anwendung in der Praxis

Lagerhaltungssystem des Hamburger Container-Hafens: Stapelung der Container entsprechend ihrer Abholung.

Animation Türme von Hanoi

Generated by Targeteam



In diesem Kapitel werden einige Klassen von Algorithmen vorgestellt, insbesondere Suchverfahren und Sortierverfahren.

- Fragestellungen des Abschnitts:
 - Welche Möglichkeiten gibt es, Datenmengen im System darzustellen?
 - Welche Möglichkeiten gibt es, in Datenmengen zu suchen?
 - Welche Möglichkeiten gibt es, Datenmengen zu sortieren?
 - Was versteht man unter der Komplexität eines Algorithmus?

Datenstrukturen

Suchverfahren

Sortierverfahren

Komplexität

Generated by Targeteam



Datenstruktur = Menge von Daten eines bestimmten Typs zusammen mit den auf der Menge ausführbaren Zugriffsoperationen.

Beispiel: natürliche Zahlen zusammen mit Grundrechenoperationen

Bei Programmierung wichtiges Hilfsmittel zur Organisation der Daten für die maschinelle Verarbeitung.

Listen

Queue - Warteschlange

Eine weitere grundlegende Datenstruktur sind Queues, die mit Warteschlangen vergleichbar sind (FIFO-Prinzip, first-in-first-out).

put() : fügt ein Element am Ende der Warteschlange hinzu

get() : entnimmt ein Element am Anfang der Warteschlange und liefert es zurück



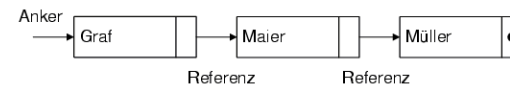
Realisierung als verkettete Liste oder als Array (falls maximale Größe bekannt).

Graphen

Generated by Targeteam



Folge von Elementen mit Reihenfolge.



Anker definiert Verweis auf 1. Element.

Referenz definiert Verweis auf nächstes Element.

Beispiele

Wort: Liste von Zeichen.

Personaldaten: Liste von Datensätzen, wobei jeder Datensatz eine Person beschreibt.

Operationen auf Listen

Erzeugen einer Liste, Einfügen eines Listenelements, Suchen eines Listenelements mit bestimmten Eigenschaften, Löschen eines Listenelements.

Reihungen (Array)

Allgemeine Listen

Generated by Targeteam



Reihungen (Array)



Liste mit fester Zahl von Elementen gleichen Typs, gemäß Reihenfolge nummeriert.

Reihungen meist in Programmiersprachen direkt unterstützt.

Deklaration durch:

```
boolean[] buffer; //deklarieren
buffer = new boolean[500]; //erzeugen
```

Generated by Targeteam



Datenstrukturen



Datenstruktur = Menge von Daten eines bestimmten Typs zusammen mit den auf der Menge ausführbaren Zugriffsoperationen.

Beispiel: natürliche Zahlen zusammen mit Grundrechenoperationen

Bei Programmierung wichtiges Hilfsmittel zur Organisation der Daten für die maschinelle Verarbeitung.

Listen

Queue - Warteschlange

Eine weitere grundlegende Datenstruktur sind Queues, die mit Warteschlangen vergleichbar sind (FIFO-Prinzip, first-in-first-out).

`put()` : fügt ein Element am Ende der Warteschlange hinzu

`get()` : entnimmt ein Element am Anfang der Warteschlange und liefert es zurück



Realisierung als verkettete Liste oder als Array (falls maximale Größe bekannt).

Graphen

Generated by Targeteam



Allgemeine Listen



einfach verkettete Listen

Listenelement: Inhalt und Referenz auf nächstes Listenelement.

doppelt verkettete Listen

Listenelement: Inhalt und Referenzen auf nächstes und vorhergehendes Listenelement.

Beispiel

```
class elem {
    elem previous;
    char content;
    elem next;
}
```

Generated by Targeteam



Graphen



Modell für Beschreibung von Objekten, die in Beziehung zueinander stehen. Menge K von Knoten (Ecken, Punkten) und Menge V von Verbindungen (Kanten). Ggf. Namen und Werte. Graph $G = (K, V)$.

gerichteter Graph

jede Kante hat Richtung.

Weg

Folge $x = K_1, K_2, \dots, K_n = y$, wobei jeweils zwischen den K_1 nach K_2 , K_2 nach K_3 , ... eine Kante besteht.

Beispiel

Straßenkarte wird durch einen Graph dargestellt:

Knoten = Städte

Kanten = Straßen zwischen den Städten

Nutzung des Graphen zur Bestimmung des kürzesten Weges zwischen 2 Städten.

Baum

spezieller Graph mit den folgenden Eigenschaften

Wurzel : ausgezeichnete Knoten, der den Ursprung des Baums kennzeichnet.

keine Zyklen.

jeder Knoten (außer der Wurzel) hat genau einen Vorgänger.

[Animation Graph Algorithmen](#)

Generated by Targeteam



Modell für Beschreibung von Objekten, die in Beziehung zueinander stehen. Menge K von Knoten (Ecken, Punkten) und Menge V von Verbindungen (Kanten). Ggf. Namen und Werte. Graph $G = (K, V)$.

gerichteter Graph

jede Kante hat Richtung.

Weg

Folge $x = K_1, K_2, \dots, K_n = y$, wobei jeweils zwischen den K_1 nach K_2 , K_2 nach K_3 , ... eine Kante besteht.

Beispiel

Straßenkarte wird durch einen Graph dargestellt:

Knoten = Städte

Kanten = Straßen zwischen den Städten

Nutzung des Graphen zur Bestimmung des kürzesten Weges zwischen 2 Städten.

Baum

spezieller Graph mit den folgenden Eigenschaften

Wurzel: ausgezeichnete Knoten, der den Ursprung des Baums kennzeichnet.

keine Zyklen.

jeder Knoten (außer der Wurzel) hat genau einen Vorgänger.

Animation Graph Algorithmen

Generated by Targeteam



In diesem Kapitel werden einige Klassen von Algorithmen vorgestellt, insbesondere Suchverfahren und Sortierverfahren.

• Fragestellungen des Abschnitts:

- Welche Möglichkeiten gibt es, Datenmengen im System darzustellen?
- Welche Möglichkeiten gibt es, in Datenmengen zu suchen?
- Welche Möglichkeiten gibt es, Datenmengen zu sortieren?
- Was versteht man unter der Komplexität eines Algorithmus?

Datenstrukturen

Suchverfahren

Sortierverfahren

Komplexität

Generated by Targeteam



TUM MMP WS 02

20 78 37 61 88

Status: Average Case Linearer Suche

gelesener Wert: - (temp)
 gesuchter Wert: 5 (keyvalue)
 Iteration: 1 (i)

```

for (int i = 0; i < array.length; i++)
  temp = array[i];
  if (temp == keyvalue)
    return i;
}
return -1;

```

Generated by Targeteam

Voraussetzung: Ordnung auf Datenelementen. Entsprechend der Ordnung in der Reihung gespeichert (aufsteigend oder absteigend).

Sei $int A[n]$ eine Reihung mit n Elementen, das aufsteigend sortiert ist.

Gesucht wird ein Element mit dem Wert x ; gesucht wird x im Bereich $A[0]$ bis $A[n-1]$.

1. wähle m zwischen 0 und $n-1$; man wird m ungefähr in der Mitte zwischen 0 und $n-1$ wählen.
2. wenn $A[m] == x$, dann sind wir fertig, gib m als Ergebnis aus.
3. wenn $x < A[m]$, dann suche weiter im Bereich $A[0]$ bis $A[m-1]$;
4. wenn $x > A[m]$, dann suche weiter im Bereich $A[m+1]$ bis $A[n-1]$

Doppelte Zahl von Elementen: ein zusätzlicher Vergleich. Bei linearer Suche: Verdopplung des Aufwandes.

Generated by Targeteam



TUM MMP WS 02

16 26 75 72 87

Status: Average Case Lineare Suche

gelesener Wert: - (temp)
 gesuchter Wert: 5 (keyvalue)
 Iteration: 1 (i)

```

for (int i = 0; i < array.length; i++)
  temp = array[i];
  if (temp == keyvalue)
    return i;
}
return -1;

```

Generated by Targeteam



Gegeben: Menge von Datensätzen. Gesucht: Datensatz mit bestimmter Eigenschaft.

Mengen von Datensätzen

Üblicherweise in Reihung oder Liste gespeichert.

Annahme für die folgenden Verfahren: Daten in einer Reihung gespeichert.

Lineare Suche

Binäre Suche

Suchverfahren Animation

Generated by Targeteam

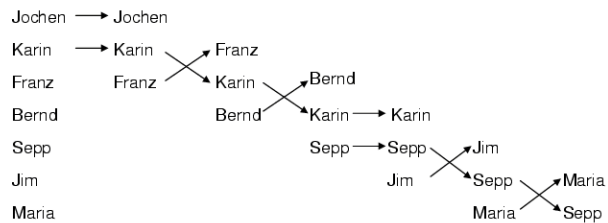


Sortieren durch Vertauschen von benachbarten Elementpaaren.

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich

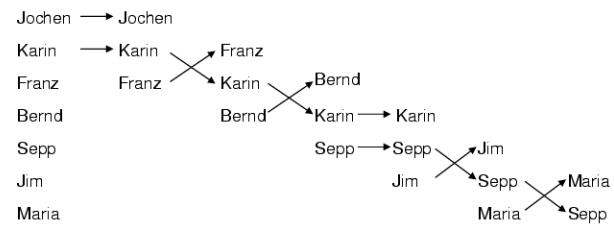


Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

Mehrere Durchgänge. Bei n Elementen maximal n Durchgänge.

Beispiel: alphabetisches Sortieren einer Menge von Namen

1. Sortierdurchgang mit jeweils paarweisen Vergleich



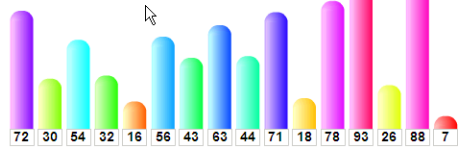
Original	nach 1	nach 2	nach 3	nach 4 Schritten
Jochen	Jochen	Franz	Bernd	Bernd
Karin	Franz	Bernd	Franz	Franz
Franz	Bernd	Jochen	Jim	Jim
Bernd	Karin	Jim	Jochen	Jochen
Sepp	Jim	Karin	Karin	Karin
Jim	Maria	Maria	Maria	Maria
Maria	Sepp	Sepp	Sepp	Sepp

Generated by Targeteam



TUM MMP WS 02

speed: medium



0 swaps 0 compares 16 bars 5 von 100 bis init