
Vorlesung 6

Ein kleines Rätsel

Wie oft muss man ein Blatt Papier (0,1 mm dick) falten, bis es eine Dicke erreicht, die der Entfernung von Erde zu Mond entspricht?

- (363.258 km=? mm)

Aufruf per Wert und Referenz

Wiederholung aus der letzten Vorlesung.

Verfestigung durch 2 Beispiele

vorlesung6_1.cpp
vorlesung6_2.cpp
alswertref.cpp

Dynamische Speicherverwaltung

Arrays

- *Syntax*

- ***new***: reservieren

```
new DATENTYP [ANZAHL_DER_ELEMENTE];
```

- ***delete***: freigeben

```
delete [] ZEIGER, _DER_DIE_ADRESSE_SPEICHERT;
```

vorlesung6_3.cpp

Dynamische Speicherverwaltung

einfache Datentypen

- *Syntax*

- ***new***: reservieren

```
new DATENTYP;
```

```
p = new int;
```

- ***delete***: freigeben

```
delete ZEIGER, _DER_DIE_ADRESSE_SPEICHERT;
```

```
delete p;
```

vorlesung6_4.cpp

Effizientes Programmieren: Algorithmen

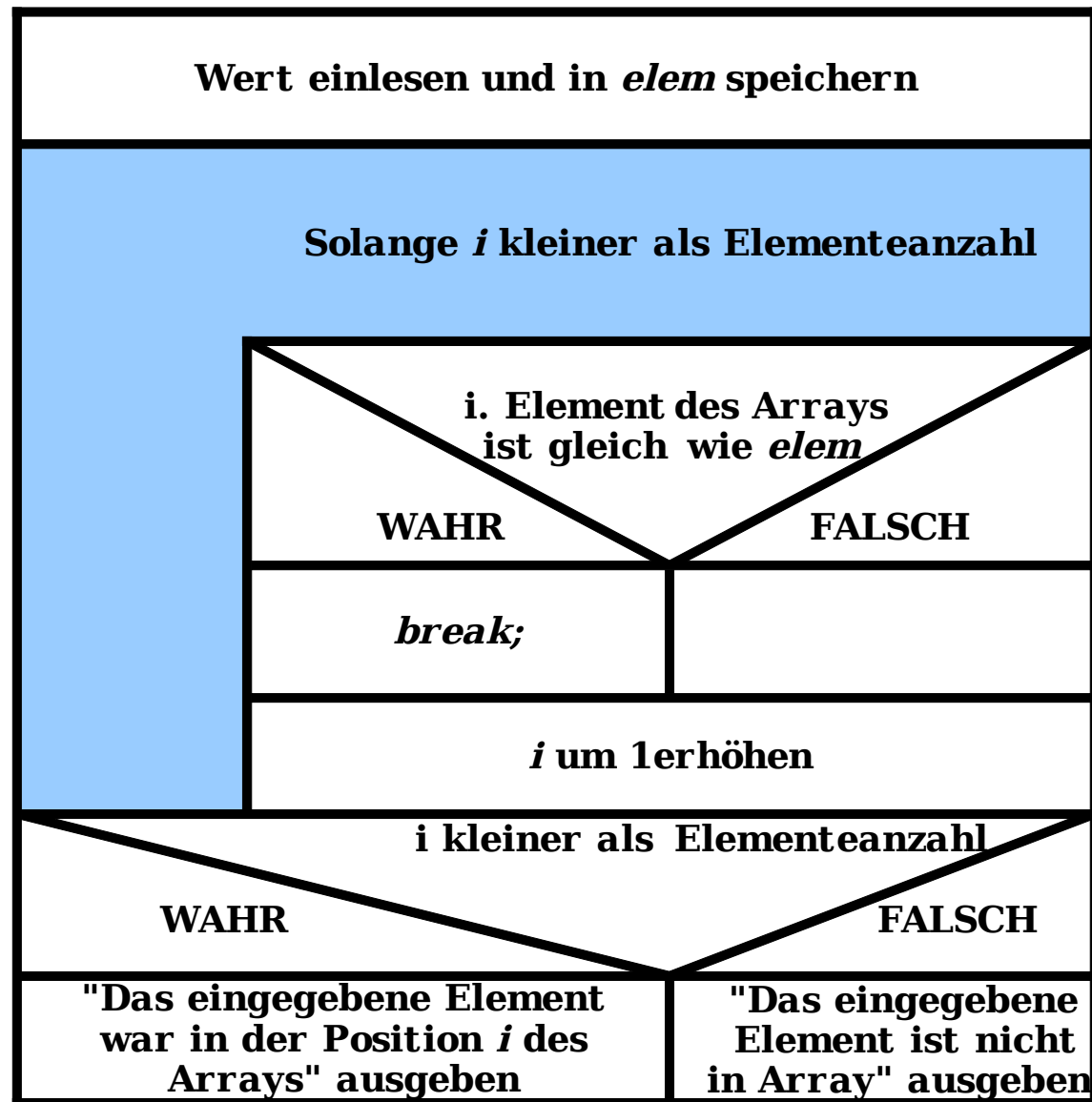
-*Algorithmen*: Verfahren, dass verwendet wird, um ein Problem durch eine endliche Anzahl von Schritten zu lösen.

Algorithmen: Darstellung

Aus dem ersten Semester:

- *Struktogramme.*
- *Flussdiagramme.*

Beispiele von Diagramme



Algorithmus: Bestandteile

Die Bestandteile ein Algorithmus sind:

- **Sachen**, die durch den Algorithmus bearbeiten werden.

Variablen

- **Operationen**, die auf den Objekten ausgeführt werden.

Operatoren-Funktionen

- **Arbeitsschritte**, in der die Operationen ausgeführt werden.

Kontrollstrukturen-Schleifen

Algorithmus - Programm

- Wenn ein Algorithmus so formuliert ist, dass Ein **Prozessor** ihn ausführen kann, wird es auch **Programm** genannt.

Algorithmen: Suche

Aufgabe:

Es wird ein Wert X in einem Feld (Array) mit n Elementen (*von 0 bis $(n-1)$*) gesucht.

20	1	-9	7	100	12	-20	1
0	1	2	3	4	5	6	7

Algorithmen (lineare Suche)

Idee

Durchlaufe A von vorn nach hinten und vergleiche jeden Wert von A mit dem gesuchten Wert s , solange bis s gefunden wird.

Algorithmen (lineare Suche)

Analyse

- Best Case: $C_{\text{best}}(n)=1$
- Worst Case: $C_{\text{worst}}(n)=n$

Algorithmen: Suche

Lösungsweg 1: die lineare Suche untersucht die Arrayelemente der Reihe nach bis das Element X gefunden ist.

20	1	-9	7	100	12	-20	1
0	1	2	3	4	5	6	7

zum Beispiel $X=12$

Aufgabe - Algorithmus - Programm

Bedürfnis, Aufgabe

Menschliche Sprache



Informatiker A

Algorithmus

Diagramm



Informatiker B

Quellcode

Programmiersprache



Compiler

Ausführbare Datei

Maschinensprache

- **Lineare Suche**

- Überprüft, ob ein Element des Arrays gleich wie das gesuchte Element ist.
- Sonst mach dasselbe mit jedem Element des Arrays.
- Das Verfahren endet, wenn entweder das Element gefunden ist oder der ganze Array durchgesucht ist.

Algorithmen (lineare Suche)

- Lineare Suche:

```
...  
element = 5; //gesuchtes Element  
  
for ( i = 0; i < n; i++ )  
{  
    if ( array [i] == element )  
        break; //Gefunden! Schleife abgebrochen.  
}  
...
```

Die Effizienz: Das Verhalten von Algorithmen bezüglich Ressourcenbedarfs:

- Rechenzeit: gesamte *Anzahl* der ***Rechenschritte***
- Speicherplatzbedarf

Algorithmus: RECHENSCHRITT

Ein Rechenschritt wird als eine grundlegende Operation des Computers betrachtet:

- Zuweisung =
- Berechnung +, %, &&, ...
- Aufruf funktion (,);
- Wert in einer Variablen (z.B. *var1*) speichern.

(rechnerische) KOMPLEXITÄT

Komplexität (Aufwand):

Das Gebiet der Informatik, das die Effizienz der Algorithmen abschätzt.

Algorithmen (lineare Suche)

- Lineare Suche:

```
...  
element = 5; //gesuchtes Element  
  
for ( i = 0; i < n; i++ )  
{  
    if ( array [i] == element )  
        break; //Gefunden! Schleife abgebrochen.  
}  
...
```

- Bei jedem Durchlauf der Schleife werden 3 Operationen ausgeführt
- Wie oft wird die Schleife ausgeführt?
 - Bester: 1 mal
 - Schlimmster: N mal
 - Durchschnitt: N/2 mal

Algorithmen (lineare Suche)

- Komplexität

- Bester: 1 mal Schleife * 3 Operationen = 3 Operationen
- Schlimmster: N mal Schleife * 3 Oper. = 3N Operationen
- Durchschnitt: N/2 mal * 3 Oper = 3N/2 Operationen

Dass heißt als Durchschnitt werden etwas **proportional** zu N
die Anzahl der Elemente

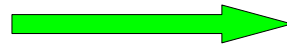
$$O(N)$$

Komplexität als eine Funktion der eingegebenen Elemente

- **$O(1)$** : konstanter Aufwand (z. B. Siehe Programm Vorlesung_5_2),
- **$O(\log n)$** : logarithmischer Aufwand (z. B. allgemeine Suchverfahren)
- **$O(n)$** : linearer Aufwand (z. B. syntaktische Programmanalyse)
- **$O(n \log n)$** : quasilinearer Aufwand (z. B. Sortierung)
- **$O(n^2)$** : quadratischer Aufwand (z. B. Vektormultiplikation)
- **$O(n^k)$** : polynomialer Aufwand (z. B. Matrizenmultiplikation (n^3)),

Algorithmen (Suche)

- Lösungsweg 2: Binäre Suche.
Vorbedingung: Sortierter Array!
 - **Mittelement** des Arrays wird überprüft
 - Falls es das Gesuchte ist, ist alles fertig
 - Sonst wird es dank der Sortierung entschieden, welche **Hälfte des Arrays** man weiter untersucht.



-9	1	7	10	12	100	200	1307
1	2	3	4	5	6	7	8

Binäre Suche: Effizienz

- **Binäre Suche.**

Größe der Eingabe: Ein Array mit n Elementen.

Die Länge des Intervalls [unten, oben] wird nach jedem Durchlauf durch 2 geteilt. Deshalb endet die Schleife nach **$\log_2(n)$** .

Deshalb ist die Effizienz proportional zum Logarithmus der Anzahl der Menge. Das heißt **$O(\log(n))$**

Der Speicher: Bereiche

- Es gibt verschiedene Bereiche im Speicher.

Der Speicher

Speicherbereiche in laufenden Programmen

ARBEITSSPEICHER

