

UNIVERSITÄT DORTMUND

Wintersemester 2007/08

**Einführung in die Informatik für
Naturwissenschaftler und Ingenieure**
(alias Einführung in die Programmierung)
(Vorlesung)

Prof. Dr. Günter Rudolph
Fachbereich Informatik
Lehrstuhl für Algorithm Engineering

Kapitel 3: Kontrollstrukturen

Inhalt

- Wiederholungen
- Auswahl (Verzweigungen)

Rudolph: EINI (WS 2007/08) • Kap. 3: Kontrollstrukturen 2

Kapitel 3: Kontrollstrukturen

Steuerung des Programmablaufes

- **Bisher: Linearer Ablauf des Programms**

```
Anweisung;
Anweisung;
Anweisung;
...
Anweisung;
```

↓

- **Oder bedingt etwas zusätzlich:**

```
Anweisung;
if ( Bedingung ) Anweisung;
Anweisung;
```

Rudolph: EINI (WS 2007/08) • Kap. 3: Kontrollstrukturen 3

Kapitel 3: Kontrollstrukturen

Steuerung des Programmablaufes: Wiederholungen

- **Die while-Schleife**

```
while ( Bedingung erfüllt ) { ← Schleifenkopf
    Anweisungen ausführen ← Schleifenrumpf
}
```

Solange die Bedingung erfüllt ist, werden die Anweisungen zwischen den geschweiften Klammern {} ausgeführt.

Danach wird hinter dem Schleifenrumpf fortgefahren.

Falls Rumpf nur eine Anweisung enthält, können Klammern {} entfallen.

Rudolph: EINI (WS 2007/08) • Kap. 3: Kontrollstrukturen 4

Steuerung des Programmablaufes: Wiederholungen

- Die while-Schleife

```
#include <iostream>
using namespace std;

int main() {
    int x = 0;
    while (x < 10) {
        cout << x << " ";
        x = x + 1;
    }
    cout << endl;
    return 0;
}
```

Ausgabe:
0 1 2 3 4 5 6 7 8 9

Steuerung des Programmablaufes: Wiederholungen

- Die while-Schleife

Achtung:
Im Schleifenrumpf sollte eine Veränderung vorkommen, die den Wahrheitswert der Bedingung im Schleifenkopf beeinflusst!

⇒ Ansonsten: Endlosschleife!

<pre>int k = 0, x = 1; while (k < 10) { x = x + 1; }</pre>	<pre>int k = 0, x = 1; while (true) { x = x + 1; }</pre>	<pre>int k = 0, x = 1; while (5) { x = x + 1; }</pre>
---	--	---

Bedingung k < 10 wird niemals false	Bedingung immer true , niemals false	Bedingung interpretiert als Konstante true
---	--	---

Steuerung des Programmablaufes: Wiederholungen

- Die while-Schleife

Ausgabe des druckbaren Standardzeichensatzes von C++ in 16er-Blöcken

```
#include <iostream>
using namespace std;

int main() {
    unsigned char c = 32;
    while (c != 0) {
        cout << c;
        c = c + 1;
        if (c % 16 == 0) cout << endl;
    }
    return 0;
}
```

Veränderung

Steuerung des Programmablaufes: Wiederholungen

- Die do/while-Schleife

```
do {
    Anweisungen ausführen
} while ( Bedingung erfüllt );
```

← Schleifenrumpf

← Schleifenfuß

Der Rechner tritt **auf jeden Fall** in den Schleifenrumpf ein, d.h. die Anweisungen zwischen den geschweiften Klammern {} werden ausgeführt.

Erst danach wird die Bedingung **zum ersten Mal** geprüft.

Solange Bedingung erfüllt ist, wird der Schleifenrumpf ausgeführt.

Danach wird hinter dem Schleifenfuß fortgefahren.

Falls der Rumpf nur eine Anweisung enthält, können Klammern {} entfallen.

Steuerung des Programmablaufes: Wiederholungen

- Die do/while-Schleife

```
#include <iostream>
using namespace std;

int main() {
    int x = 0;
    do {
        cout << x << " ";
        x = x + 1;
    } while (x < 10);
    cout << endl;
    return 0;
}
```

Achtung!
Weil wir Anfangswert von **x** und Bedingung kennen, wissen wir, dass der Rumpf mindestens einmal durchlaufen wird!
Das ist nicht immer so!

Ausgabe:
0 1 2 3 4 5 6 7 8 9

Steuerung des Programmablaufes: Wiederholungen

- Die while-Schleife als do/while-Schleife

```
while (Bedingung erfüllt) {
    Anweisungen ausführen;
}
```



```
if (Bedingung erfüllt) {
    do {
        Anweisungen ausführen;
    } while (Bedingung erfüllt);
}
```

Steuerung des Programmablaufes: Wiederholungen

- Die do/while-Schleife als while-Schleife

```
do {
    Anweisungen ausführen;
} while (Bedingung erfüllt);
```



```
Anweisungen ausführen;
while (Bedingung erfüllt) {
    Anweisungen ausführen;
}
```

Steuerung des Programmablaufes: Wiederholungen

- Wann ist die do/while-Schleife sinnvoll?

Wenn wir zur Zeit der Programmerstellung wissen, dass der Schleifenrumpf **mindestens einmal** durchlaufen werden muss!



```
int n;
do {
    cout << "Anzahl Sterne (1-8): ";
    cin >> n;
    while (n < 1 || n > 8);
    while (n--) cout << '*';
}
```

Verlangt Eingabe einer Zahl so lange bis der Wert zwischen 1 und 8 ist

Kurzschreibweise:
Gibt Wert von **n** (hier an Bedingung), erniedrigt dann **n** um 1

Steuerung des Programmablaufes: Wiederholungen

- Exkurs: Kurzschreibweisen für Inkrement / Dekrement

<code>x = ++k;</code>	entspricht	<code>k = k + 1; x = k;</code>
<code>x = k++;</code>	entspricht	<code>x = k; k = k + 1;</code>
<code>x = --k;</code>	entspricht	<code>k = k - 1; x = k;</code>
<code>x = k--;</code>	entspricht	<code>x = k; k = k - 1;</code>

Steuerung des Programmablaufes: Wiederholungen

- Exkurs: Kurzschreibweisen für Inkrement / Dekrement

<code>while (--k) {</code> Anweisung; <code>}</code>	entspricht	<code>k = k - 1;</code> <code>while (k) {</code> Anweisung; <code>k = k - 1;</code> <code>}</code>
<code>while (k--) {</code> Anweisung; <code>}</code>	entspricht	<code>while (k) {</code> <code>k = k - 1;</code> Anweisung; <code>}</code> <code>k = k - 1;</code>

(analog für `++k` und `k++`)

Steuerung des Programmablaufes: Wiederholungen

- Exkurs: Kurzschreibweisen für Zuweisungen

<code>k += 5;</code>	entspricht	<code>k = k + 5;</code>
<code>k -= j-1;</code>	entspricht	<code>k = k - (j-1);</code>
<code>k *= i+2;</code>	entspricht	<code>k = k * (i+2);</code>
<code>k /= i*2-1;</code>	entspricht	<code>k = k / (i*2-1);</code>
<code>k %= 16;</code>	entspricht	<code>k = k % 16;</code>
<code>k = i = j = 1;</code>	entspricht	<code>k = (i = (j = 1));</code>

Steuerung des Programmablaufes: Wiederholungen

- Die `for` - Schleife

```
for ( Initialisierung; Bedingung; Veränderung ) {
    Anweisungen ausführen;
}
```

← Schleifenkopf

← Schleifenrumpf

Bei der **Initialisierung** wird Startwert des Schleifenzählers festgelegt.
Die **Bedingung** prüft, ob Endwert des Schleifenzählers noch nicht erreicht ist.
Mit der **Veränderung** wird die Bedingung beeinflusst.

Steuerung des Programmablaufes: Wiederholungen

• Die for - Schleife

```
for ( Initialisierung; Bedingung; Veränderung ) {
    Anweisungen ausführen;
}
```

1. Zuerst wird der Schleifenzähler initialisiert.
2. Falls Bedingung erfüllt:
 - a) Führe Anweisungen aus.
 - b) Führe Veränderung aus.
 - c) Weiter mit 2.
3. Falls Bedingung nicht erfüllt: Fahre nach Schleifenrumpf fort.

Steuerung des Programmablaufes: Wiederholungen

• Die for - Schleife: Beispiele

A) `for (k = 0; k < 10; k++) cout << k << ' ';`

Ausgabe:
0 1 2 3 4 5 6 7 8 9

B) `for (k = 0; k < 10; k += 2) cout << k << ' ';`

Ausgabe:
0 2 4 6 8

Steuerung des Programmablaufes: Wiederholungen

• Die for - Schleife: Beispiele

C) `float x;
for (x = 0.0; x <= 3.0; x += 0.1)
 cout << x << ": " << x*x << endl;`

D) `enum tagT { MO, DI, MI, DO, FR, SA, SO };
tagT tag;
int at = 0;
for (tag = MO; tag <= FR; ++tag) at++;
cout << "Arbeitstage: " << at << endl;`

Steuerung des Programmablaufes: Wiederholungen

• Die for - Schleife:

Initialisierung, Bedingung, Veränderung sind optional!

`int i = 9;
for (; i >= 0; i--) cout << i << " ";`

`int i = 10;
for (; --i >= 0;) cout << i << " ";`

`int i = 10;
for (; i > 0;) { i--; cout << i << " "; }`

`int i = 10;
for (; ;) cout << i << " ";`

identische Ausgabe:
Ziffern 9 bis 0
abwärts



Steuerung des Programmablaufes: Wiederholungen

• Die break – Anweisung (Teil 1)

Alternative Beendigungen von Schleifen:

```
for (i = 0; ; ) {
  cout << i << " ";
  if (i > 9) break;
  i++;
}
```

Die break – Anweisung unterbricht die Schleife sofort!
Es wird direkt hinter dem Schleifenrumpf fortgefahren!
Das funktioniert auch in Schleifenrumpfen von while und do/while!

Steuerung des Programmablaufes: Wiederholungen

• Die break – Anweisung (Teil 1)

Alternative Beendigungen von Schleifen:

```
for (i = 0; i < 10; i++) {
  cout << i << ": ";
  for (j = 0; j < 10; j++) {
    if (i + j >= 5) break;
    cout << j << " ";
  }
  cout << endl;
}
```

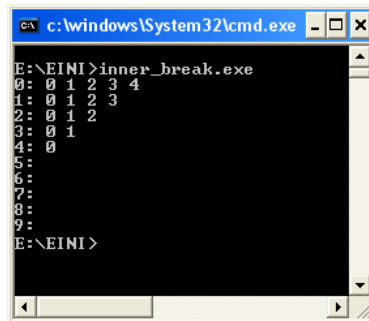


Die break – Anweisung unterbricht **nur** die **aktuelle** Schleife sofort!

Steuerung des Programmablaufes: Wiederholungen

• Die break – Anweisung (Teil 1)

Ausgabe:



Die break – Anweisung unterbricht **nur** die **aktuelle** Schleife sofort!

Steuerung des Programmablaufes: Wiederholungen

• Die berüchtigte goto – Anweisung: goto Bezeichner;

Alternative Beendigungen von Schleifen:

```
for (i = 0; ; ) {
  cout << i << " ";
  if (i > 9) goto marke;
  i++;
}
marke: cout << "Schleife beendet!";
```



Bei der goto – Anweisung wird sofort zur angegeben Markierung gesprungen!
Es wird direkt bei der Markierung fortgefahren!
Das funktioniert auch in Schleifenrumpfen von while und do/while!

Die Verwendung von goto ist niemals notwendig! Unbedingt vermeiden!

Steuerung des Programmablaufes: Wiederholungen

- Die `continue` – Anweisung:

Erzwingt einen sofortigen Sprung an das Schleifenende!
Nur der **aktuelle** Schleifendurchlauf wird beendet,
nicht die ganze Schleife (wie bei `break`)!

```
for ( i = 0; i < 10; i++ ) {  
  Anweisungen ausführen;  
  if (Bedingung) continue;  
  Anweisungen ausführen;  
}
```

Das funktioniert auch in Schleifenrumpfen von `while` und `do/while`!

Steuerung des Programmablaufes: Wiederholungen

- Die `continue` – Anweisung:

Ermöglicht manchmal besser lesbaren / nachvollziehbaren Programmcode.
Ist niemals wirklich notwendig.

```
for ( i = 0; i < 10; i++ ) {  
  Anweisungen ausführen;  
  if (Bedingung) continue;  
  Anweisungen ausführen;  
}
```



```
for ( i = 0; i < 10; i++ ) {  
  Anweisungen ausführen;  
  if (!Bedingung) Anweisungen ausführen;  
}
```

Steuerung des Programmablaufes: Wiederholungen

- Die `for` – Schleife als `while` – Schleife:

```
for ( Initialisierung; Bedingung; Veränderung ) {  
  Anweisungen ausführen;  
}
```



```
Initialisierung;  
while ( Bedingung ) {  
  Anweisungen ausführen;  
  Veränderung;  
}
```

Steuerung des Programmablaufes: Wiederholungen

- Die `while` – Schleife als `for` – Schleife:

```
while ( Bedingung ) {  
  Anweisungen ausführen;  
}
```



```
for ( ; Bedingung; ) {  
  Anweisungen ausführen;  
}
```

Steuerung des Programmablaufes: Auswahl

- Einseitige Auswahl: `if`

`if (Bedingung) Anweisung;` nur eine Anweisung ausführen

```
if (Bedingung) {
    Anweisung;
    Anweisung;
    ...
    Anweisung;
}
```

einen ganzen **Block** von Anweisungen ausführen

Wenn die Bedingung erfüllt ist, dann wird die Anweisung oder der Block von Anweisungen ausgeführt, sonst eben nicht!

Steuerung des Programmablaufes: Auswahl

- Zweiseitige Auswahl: `if else`

```
if (Bedingung)
    Anweisung1;
else
    Anweisung2;
```

wenn Bedingung erfüllt, dann Anweisung1 ausführen, ansonsten Anweisung2 ausführen!

```
if (Bedingung) {
    Anweisungsblock1;
}
else {
    Anweisungsblock2;
}
```

Achtung! Hier kein Semikolon hinter der Klammer } erlaubt!

Steuerung des Programmablaufes: Auswahl

- Zweiseitige Auswahl: `if else`

Beispiel:

```
if (kunde.umsatz >= 100000) {
    kunde.bonus = 5000;
    kunde.skonto = 0.03;
    kunde.status = GuterKunde;
}
else {
    kunde.bonus = 2000;
    kunde.skonto = 0.02;
    kunde.status = NormalerKunde;
}
```

```
enum StatusT = {
    GuterKunde,
    NormalerKunde,
    SchlechterKunde
};
```

```
struct KundeT {
    int umsatz;
    int bonus;
    float skonto;
    statusT status;
};
```

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: `if else - Schachtelung (nesting)`

```
if ( Bedingung1 ) Anweisung1;
else
    if ( Bedingung2 ) Anweisung2;
    else
        if ( Bedingung3 ) Anweisung3;
        else Anweisung4;
```

Achtung!
Festlegung: Das **letzte else** bezieht sich auf das **letzte if!**
Erfordert Logik einen anderen Bezug, dann Klammern { } setzen!

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: `if else` - Schachtelung (*nesting*)

```
if ( Bedingung0 )
  if ( Bedingung1 ) Anweisung1;
  else
    Anweisung2;
```

else bezieht sich auf Bedingung1

```
if ( Bedingung0 ) {
  if ( Bedingung1 ) Anweisung1;
}
else
  Anweisung2;
```

else bezieht sich auf Bedingung0

↓ äquivalent, aber ohne Klammern:

```
if ( !Bedingung0 ) Anweisung2;
else if ( Bedingung1 ) Anweisung1;
```

B0	B1	
F	F	A2
F	T	A2
T	F	-
T	T	A1

F: false
T: true

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: `if else` - Schachtelung (*nesting*)

```
if ( a > b ) cout << "a > b";
if ( a < b ) cout << "a < b";
if ( a == b ) cout << "a == b";
```

ohne Schachtelung:
immer 3 Vergleiche!

```
if ( a > b ) cout << "a > b";
else
  if ( a < b ) cout << "a < b";
  else cout << "a == b";
```

mit Schachtelung:
1 oder 2 Vergleiche!

⇒ Effizienzsteigerung: Schnelleres Programm!

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: `switch`

```
switch (Ausdruck) {
  case c1: Anweisungen_1; break;
  case c2: Anweisungen_2; break;
  ...
  case cn: Anweisungen_n; break;
  default: Anweisungen;
}
```

Der Ausdruck muss einen abzählbaren Datentyp ergeben:
`char, short, int, long, enum, bool (false < true)`

Konstanten `c1` bis `cn` müssen paarweise verschieden sein!

Ist `Ausdruck == Wert` einer Konstanten, dann werden Anweisungen bis `break` ausgeführt; sonst Anweisungen von `default`.

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: `switch` ohne `default`

```
switch (Ausdruck) {
  case c1: Anweisungen_1; break;
  case c2: Anweisungen_2; break;
  ...
  case cn: Anweisungen_n; break;
}
```

`default` - Zweig kann entfallen ⇒

besser: `default` mit leerer Anweisung!

noch besser: `default` mit leerer Anweisung und Kommentar!

Weglassen nur selten sinnvoll, z.B. bei `enum` (alle Werte werden unterschieden).
Oder bei `bool` (nur 2 Werte), wo `if`-Anweisungen einfacher wären.

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: switch ohne default

```
switch (Ausdruck) {
  case c1: Anweisungen_1; break;
  case c2: Anweisungen_2; break;
  ...
  case cn: Anweisungen_n; break;
  default: ; // leere Anweisung
}
```

leere Anweisung Kommentar

Exkurs: Kommentare

Ein Kommentar im Programmtext

- dient der Kommentierung / Erklärung / Beschreibung des Programmcodes
- wird vom Compiler ignoriert

Nur in C++:

```
int a = 1; // Kommentar      ignoriert wird ab // bis zum Ende der Zeile
a = a + 3;
```

In C und C++:

```
int a = 1; /* Kommentar:      ignoriert werden alle Zeichen zwischen
blablabla blabla */          /* und */, auch über mehrere Zeilen!
a = a + 3;
```

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: switch mit „fehlenden“ break's

```
switch (Ausdruck) {
  case c1:
  case c2: Anweisungen_2; break;
  case c3: Anweisungen_3;
  case c4: Anweisungen_4; break;
  case c5: Anweisungen_5; break;
  default: Anweisungen;
}
```

break führt zum Verlassen der switch – Anweisung!

Fehlt am Ende eines case-Zweiges ein break, dann werden Anweisungen der nachfolgenden case-Zweige ausgeführt bis auf ein break gestossen wird.

Steuerung des Programmablaufes: Auswahl

- Mehrfache Auswahl: switch

Beispiel: Abfrage, ob weiter gemacht werden soll; Eingabe nur j, J, n oder N

```
char c;                    // einzulesendes Zeichen
bool OK;                  // true, falls Eingabe in {j,J,n,N}
bool weiter;              // true, falls weiter gemacht wird
do {
  cin >> c;
  switch (c) {
    case 'j':
    case 'J': OK = weiter = true; break;
    case 'n':
    case 'N': OK = true; weiter = false; break;
    default : OK = false;
  }
} while (!OK);
```