6. Kontrollfluss

Prof. Dr. Markus Gross

Informatik I für D-ITET (WS 03/04)

- Die **`for`** Schleife
- Inkrement und Dekrement Operatoren **`++`**, **`--`**
- Vergleichsoperatoren **`<`**, **`>`**, **`>=`**, **`==`**
- Die **`while`** Schleife
- Die **`do while`** Schleife
- Mehrfachschleifen
- **`enum`** Typ

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Kontrollanweisungen

- Eine *Kontrollanweisung* beeinflusst den Programmablauf gezielt und datenabhängig
  - *Schleifen* für sich wiederholende Berechnungen (*loops*)
  - Bedingte *Sprünge* (*Conditional branches*)
- In C++ gibt es verschiedene Schleifen
  - `for`
  - `while`
  - `do while`
- Bedeutung ist recht ähnlich
- Oft auch eine Frage des Programmierstils

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_1: for-Schleife

```
// forloop.cpp -- introducing the for loop
#include <iostream>
using namespace std;
int main()
{
    int i; // create a counter
    //   initialize; test ; update
    for (i = 0; i < 5; i++)
        cout << "C++ knows loops.\n";
    cout << "C++ knows when to stop.\n";
    return 0;
}
```

Definition eines Index

Initialisierung

Test

Inkrement des Index

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

`for`-Schleife

- Einfachstes Konstrukt um Anweisungen zu wiederholen
- Syntax recht einfach:

```
int i;
for (i=0; i<5; i++)
   cout << "Hi\n";
```

- Zunächst *Schleifenindex* definieren (`int i;`)
- *Initialisierung* erfolgt in der Schleife (`i=0;`)
- *Abbruchbedingung* ebenfalls (`i<5;`)
  - Schleife wird abgearbeitet solange `i<5`
  - Bei `i==5` erfolgt Abbruch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

`for`-Schleife

- *Update* des Schleifenindex durch Inkrementoperator (`i++;`)
  - `i++;` entspricht `i=i+1;`
- Allgemeiner Ablauf einer `for`-Schleife:
  - Initialisierung des Index (nur einmal ausgeführt)
  - Test auf Abbruchbedingung
  - Ausführung der Anweisungen in der Schleife (Block)
  - Update des Index
- Initialisierung, Test und Update sind in Klammern eingeschlossen und durch Semikolon getrennt

```
for (Initialisierung; Test-Ausdruck; Update)
    Body; // Anweisungen im Block
```

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

`for`-Schleife

- Der Testausdruck kann beliebig komplex sein
- Beispiel:

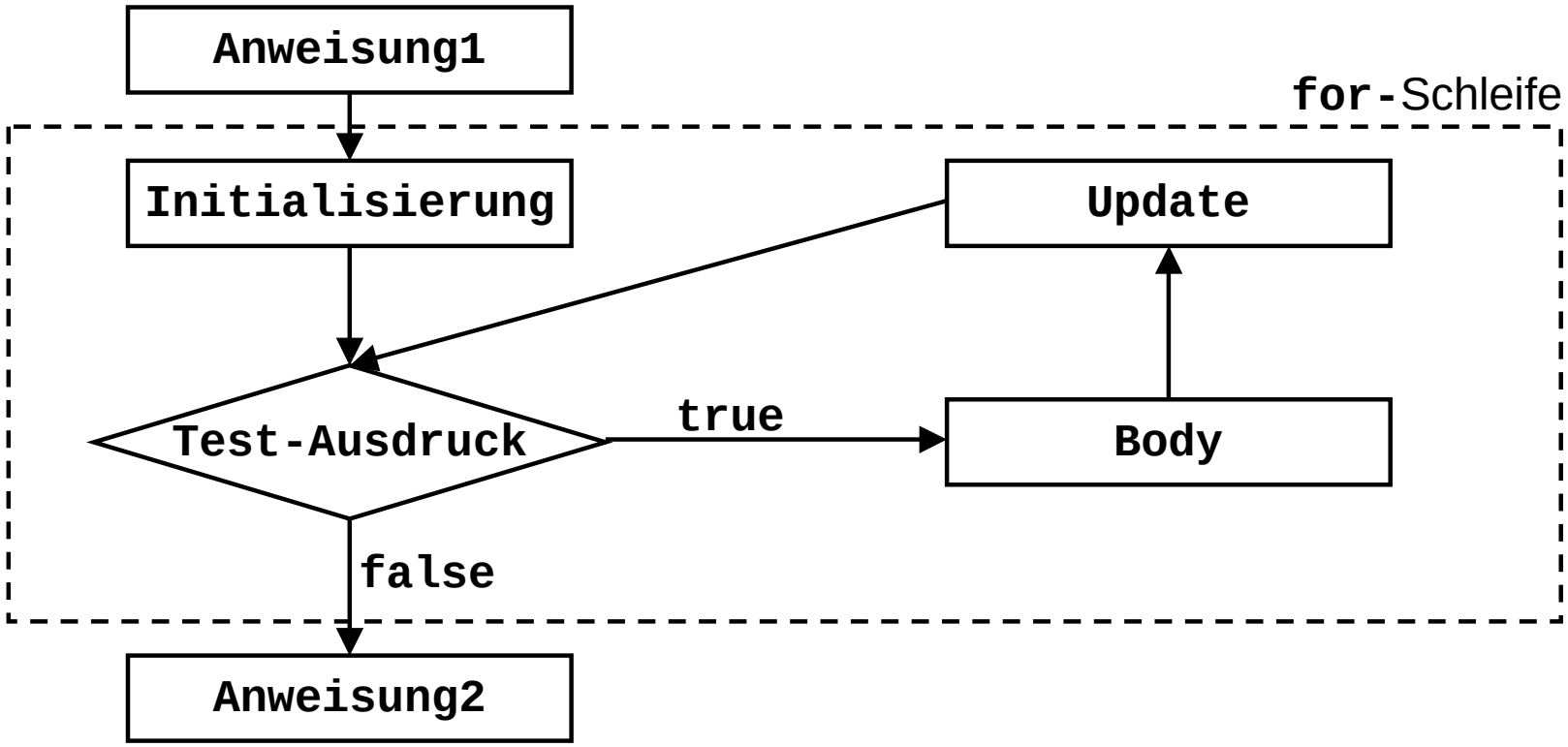
```
int limit;
cin >> limit;
for (i=limit; i; i--) // i == true
```

- Der Ausdruck `i` wird als boolean evaluiert
- Dekrementoperator (`i--;`)
- Der Test in einer `for`-Schleife wird VOR Abarbeitung der Schleifenanweisungen durchgeführt
- Wenn Testbedingung nicht erfüllt, dann Abbruch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bild `for`-Schleife

```
Anweisung1;
for (Initialisierung; Test-Ausdruck; Update)
        Body;
Anweisung2;
```

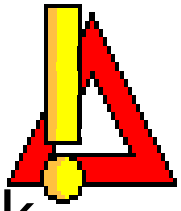


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Spezialitäten der `for`-Schleife

- Definition des Index auch wie folgt möglich:

```
for (int i=limit; i; i--) //scope !!
```



- Gültigkeit der Variable auch NACH Schleifenblock
- Inkremente können beliebige Grössen haben

```
int by = 5;
for (int i=0; i<100; i=i+by)
```

- Auch Funktionen können zur Initialisierung verwendet werden
- Bei mehreren Anweisungen innerhalb der Schleife muss die Blockklammer `{ }` verwendet werden
- Schleifen auch über **`char`** Indizes möglich

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_2: Strings

```
// compstr.cpp -- comparing strings
#include <iostream>
#include <cstring>     // prototype for strcmp()
using namespace std;

int main()
{
    char word[5] = "?ate";

    for (char ch = 'a'; strcmp(word, "mate"); ch++)
    {
        cout << word << "\n";
        word[0] = ch;
    }
    cout << "After loop ends, word is " << word << "\n";
    return 0;
}
```

Initialisierung

strcmp-Funktion true?

Index vom Typ char

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Spezialitäten der `for`-Schleife

- Beispiel: Block

```
for (int i=0; ; i++) // i == true
{
   alpha = beta;
   gamma = delta;
}
```

- Formatierung mittels Tabulatoren
- Variablen innerhalb von Blöcken haben *Block Scope*
- Gleichnamige Variablen werden zwischengespeichert und wieder rekonstruiert
- Komma-Operator erlaubt ebenfalls die Auswertung mehrerer Ausdrücke

```
for (int i=0; i<100; i--, j++)
```

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bsp.: Block-Scope von Variablen

```
int main(void)
{
   int x = 20;
   {
       cout << x; // newline omitted
       int x = 10;
       cout << x;
   }
   cout << x;
   return 0;
}
```

`while`-Schleife

- Stellt eine abgespeckte `for`-Schleife dar
  - Keine Initialisierung und kein Update
- Allgemeine Form:

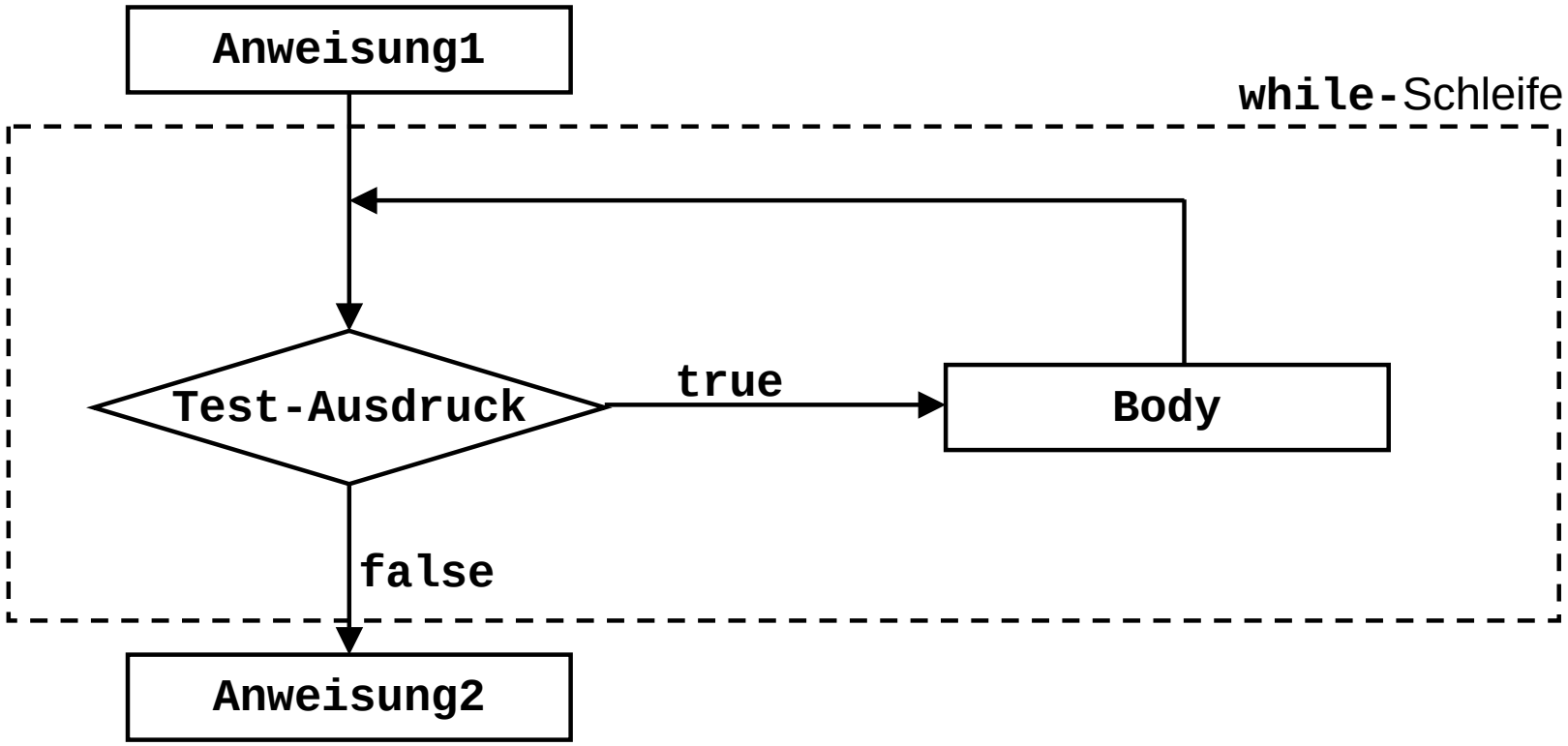
```
while (Test-Ausdruck)
    Body; // Anweisungen im Block
```

- Wiederholt, solange Testbedingung `true`
- Wenn Anweisungen in Body abgearbeitet, wird Testbedingung neu evaluiert
- Schleife beendet, wenn Test-Ausdruck `true`
- Kann zu Endlosschleifen führen
- Test-Audruck muss im Body verändert werden
  - `while (true)`

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bild `while`-Schleife

```
Anweisung1;
while (Test-Ausdruck)
        Body;
Anweisung2;
```



Beispiel_3: Strings mit while

```
// while.cpp -- introducing the while loop
#include <iostream>
using namespace std;

const int ArSize = 20;

int main()
{
    char name[ArSize];

    cout << "Your first name, please: ";
    cin >> name;
    cout << "Here is your name, verticalized and ASCIIized:\n";
    int i = 0;                      // start at beginning of string
    while (name[i] != '\0')         // process to end of string
    {
        cout << name[i] << ": " << int(name[i]) << '\n';
        i++;                        // don't forget this step
    }
    return 0;
}
```

Initialisierung

Testbedingung '\0'

`for` versus `while`

- Beide Schleifentypen können ineinander übergeführt werden

```
for (Initialisierung; Test; Update)
{
   Anweisungen;
}
```

  - Ist äquivalent zu

```
Initialisierung;
while (Test)
{
   Anweisungen;
   Update;
}
```

`for` versus `while`

- Ebenso

```
while (Test)
{
   Anweisungen;
}
```

  - Ist äquivalent zu

```
for ( ;Test; )
{
   Anweisungen;
}
```

- Endlosschleife
  - `for ( ; ; ) // entspricht while (true)`

`do while`-Schleife

- Grundsätzlicher Unterschied zu **`for`** und **`while`**
- Schleifenbedingung wird erst NACH Ausführung der Anweisungen im Body geprüft
- Body wird also MINDESTENS einmal ausgeführt
- Allgemeine Form:

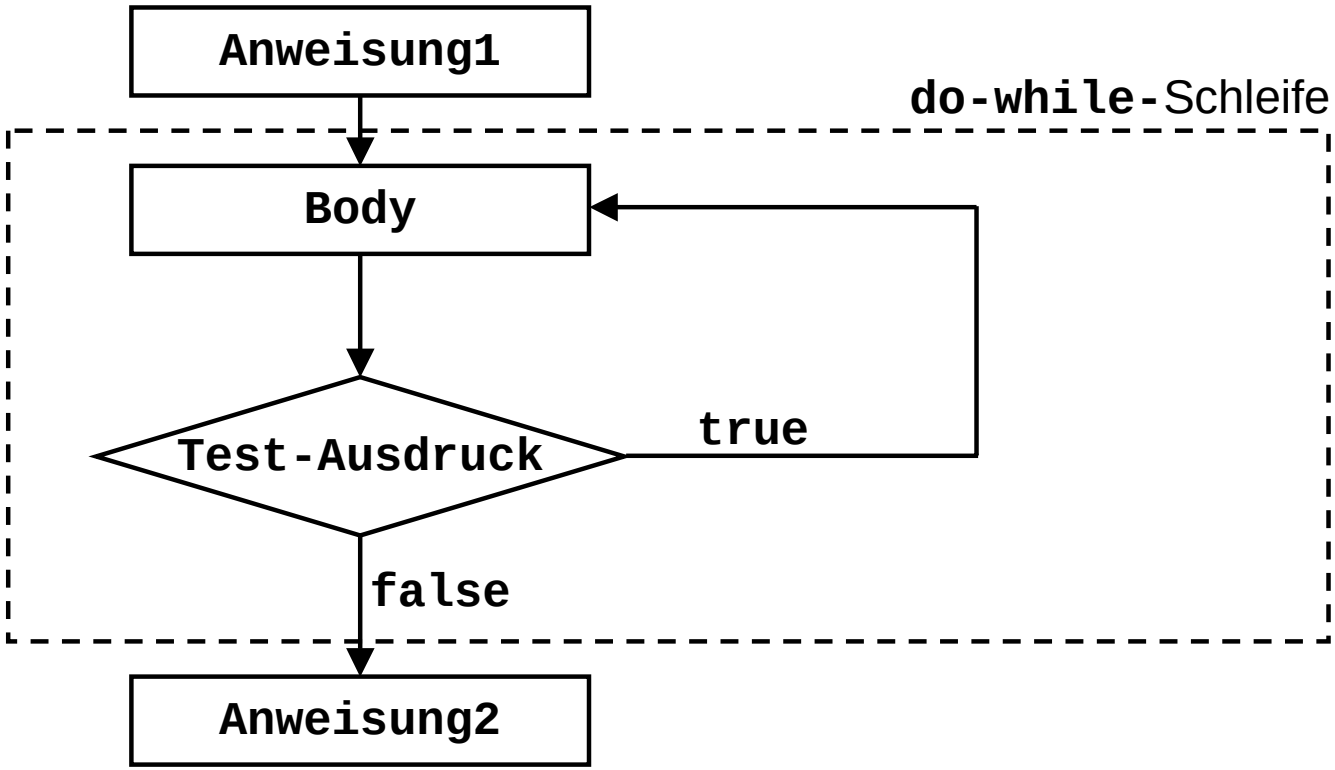
```
do
   Body; // Anweisungen im Block
while (Test-Ausdruck);
```

- Sinnvoll bei Eingabe verwendbar
- Allgemein sind **`for`** und **`while`** sicherer

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bild do while-Schleife

```
Anweisung1;
do
        Body;
while (Test-Ausdruck)
Anweisung2;
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Beispiel_4: Eingabe mit do while

```
// dowhile.cpp -- exit-condition loop
#include <iostream>
using namespace std;

int main()
{
    int n;

    cout << "Enter numbers in the range 1-10 to find ";
    cout << "my favorite number\n";
    do
    {
        cin >> n;       // execute body
    } while (n != 7);   // then test
    cout << "Yes, 7 is my favorite.\n" ;
    return 0;
}
```

Eingabe von n

Test von n

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Mehrfachschleifen und 2D-Arrays

- Schleifen können ineinander verschachtelt werden
- Wichtige Anwendung ist die Adressierung von 2D Arrays
- Statische 2D Arrays werden intern als Arrays von 1D Arrays abgespeichert
- Dynamische 2D Arrays müssen explizit mit **new** alloziert werden
- Es bedarf eines weiteren Arrays von Pointern
- Adressierung über Pointer-Arithmetik

Beispiel_5: Verschachtelte Schleife

```
// nested.cpp -- nested loops and 2-D array
#include <iostream>
using namespace std;
const int Cities = 5;
const int Years = 4;
int main()
{
    const char cities[Cities][128] =  // array of 5 strings
    {
        "Gribble City",
        "Gribbleton",
        "New Gribble",
        "San Gribble",
        "Gribble Vista"
    };

    int maxtemps[Years][Cities] =  // 2-D array
    {
        {94, 98, 87, 103, 101}, // values for maxtemps[0]
        {98, 99, 91, 107, 105}, // values for maxtemps[1]
        {93, 91, 90, 101, 104}, // values for maxtemps[2]
        {95, 100, 88, 105, 103} // values for maxtemps[3]
    };

    cout << "Maximum temperatures for 1995 - 1998\n\n";

    for (int city = 0; city < Cities; city++)
    {
        cout << cities[city] << ":\t";
        for (int year = 0; year < Years; year++)
            cout << maxtemps[year][city] << "\t";
        cout << "\n";
    } return 0;
}
```

Array von Strings

Initialisierung

Statisches 2D Array

Verschachtelte Schleife

Guter Programmierstil

- Der **`enum`**-Typ kann elegant in Verbindung mit der **`switch`**-Anweisung verwendet werden
- Hierbei legt **`enum`** den Wertebereich der Integer-Labels fest
- Labels können dann symbolisch (mit Namen) verwendet werden

Beispiel: switch und enum

```
// enum.cpp -- use enum
#include <iostream>
using namespace std;
// create named constants for 0 - 6

enum {red, orange, yellow, green, blue, violet, indigo};

int main()
{
    cout << "Enter color code (0-6): ";
    int code;
    cin >> code;
    while (code >= red && code <= indigo)
    {
        switch (code)
        {
            case red    : cout << "Her lips were red.\n"; break;
            case orange : cout << "Her hair was orange.\n"; break;
            case yellow : cout << "Her shoes were yellow.\n"; break;
            case green  : cout << "Her nails were green.\n"; break;
            case blue   : cout << "Her sweatsuit was blue.\n"; break;
            case violet : cout << "Her eyes were violet.\n"; break;
            case indigo : cout << "Her mood was indigo.\n"; break;
        }
        cout << "Enter color code (0-6): ";
        cin >> code;
    }
    cout << "Bye\n";
    return 0;
}
```

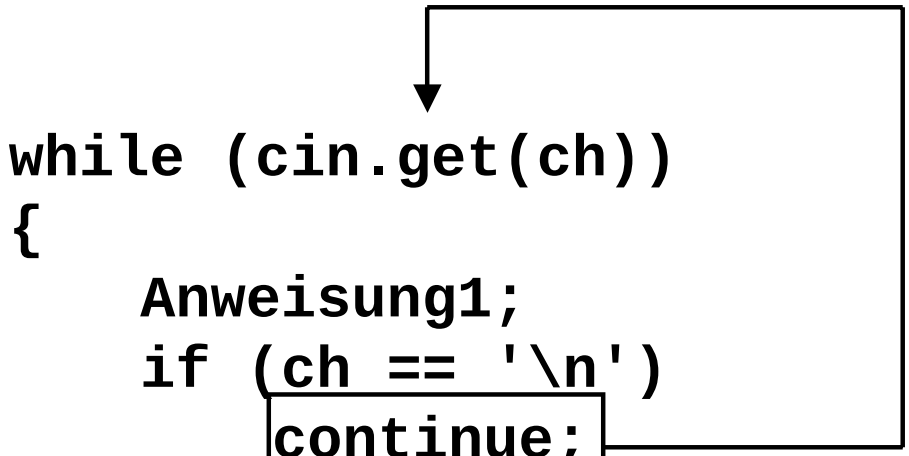
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Abbruch und Weiterführung

- **`break`** kann sowohl mit **`switch`** als auch in Schleifen verwendet werden
- Die **`break`** Anweisung bewirkt einen "Ausbruch" aus der aktuellen Kontollstruktur
  - Schleife wird verlassen
- Programmfluss springt zu erster Anweisung nach Kontrollstruktur
- **`continue`** überspringt den Rest der Kontrollstruktur
  - Sprung zum Ende des Schleifenblocks und erneutes Abprüfen der Schleifenbedingung
- **`break`** und **`continue`** sind *unbedingte Sprünge*

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bild: `continue`



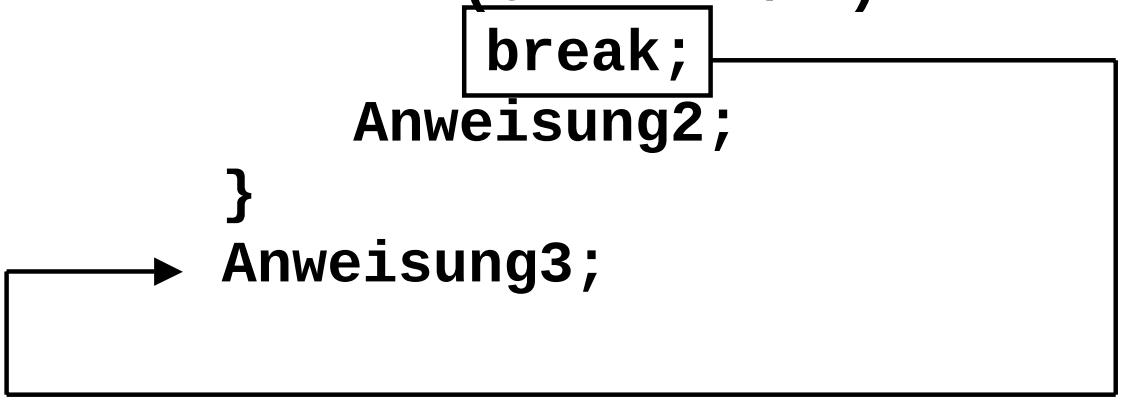
```
while (cin.get(ch))
{
    Anweisung1;
    if (ch == '\n')
        continue;
    Anweisung2;
}
Anweisung3;
```

`continue` überspringt den Rest der Kontrollstruktur: Sprung zum Ende des Schleifenblocks und erneutes Abprüfen der Schleifenbedingung

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bild: break

```
while (cin.get(ch))
{
    Anweisung1;
    if (ch == '\n')
        break;
    Anweisung2;
}
Anweisung3;
```



break bewirkt einen "Ausbruch" aus der aktuellen Kontollstruktur: Schleife wird verlassen