Herzlich willkommen!

Dozent: Dipl.-Ing. Jürgen Wemheuer

Teil 4: Schleifenkonstruktionen

Mail: wemheuer@ewla.de
Online: http://cpp.ewla.de/

Schleifenkonstruktion goto (veraltet!)

- Sprungmarke setzen der Form „Label:“
- Zu dieser Sprungmarke springen per goto-Anweisung:

```
int zaehler = 0;
Label: zaehler++;
       cout << "Zaehler: " << zaehler << "\n";
       if (zaehler < 5) goto Label;
       cout << "Ende Zaehler: " << zaehler << "\n";
```

Funktioniert zwar, macht man aber nicht mehr, weil:

- führt bei unkontrollierter Anwendung zu „Spaghetti-Code“
- Sprünge über weite Distanzen werden ineffizient
- wir haben ja strukturierte for-, while- und do…while-Schleifen

Schleifenkonstruktion mit while


```
while (!sleep) {
    ++sheep;
}
```

Schleifenkonstruktion while-Schleife 1

Eine Folge von Anweisungen wird sooft wiederholt, solange die Test- oder Startbedingung der while-Schleife gleich **true** ist.
Die Folge von Anweisungen bildet einen Block und wird als Verbundanweisung in {Block} eingeschlossen:

```
int zaehler = 0;
while (zaehler < 5) {
      zaehler++;
      cout « "Zaehler: " « zaehler « "\n";
}
cout « "Ende Zaehler: " « zaehler « "\n";
```

Die while-Anweisung kann als Testbedingung jeden beliebigen C++-Ausdruck beinhalten, Beispiel:

```
while ((Laenge<kurz && Breite<schmal) || Anzahl<Stueck)
{ Block von Anweisungen }
```

Schleifenkonstruktion while-Schleife 2

Die Schleifenbedingung wird bereits am Anfang geprüft, es kann also sein, dass die Schleife gar nicht ausgeführt wird:

```
int Bedingung = 0;
while (Bedingung > 0) {
      cout << "Hallo Schleife";
      Bedingung -= 2;
}
cout << "Goodbye Schleife";
```

Außerdem muss unbedingt im Schleifenblock die Bedingung behandelt werden:

```
int Bedingung = 10;
while (Bedingung > 0) {
cout << "Hier bin ich gefangen...!!!"; }
```

do...while-Schleife

Bei einer do...while-Schleife wird der Anweisungsblock ohne Prüfung einmal durchlaufen. Erst am Ende wird die Bedingung zum erneuten Ausführen der Schleifenanweisung geprüft:

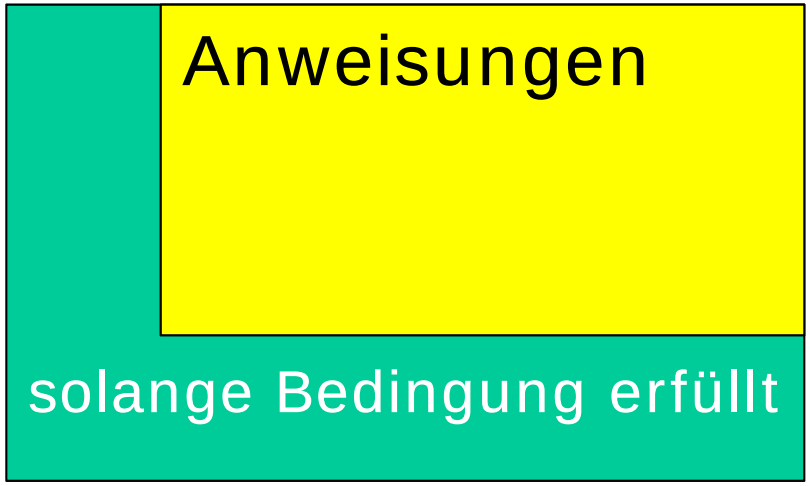
```
int Bedingung = 0;
do {
        cout << "Hallo Schleife";
        Bedingung -= 2;
} while (Bedingung > 0);
cout << "Goodbye Schleife";
```

Natürlich gibt es auch do...while-Endlosschleifen:

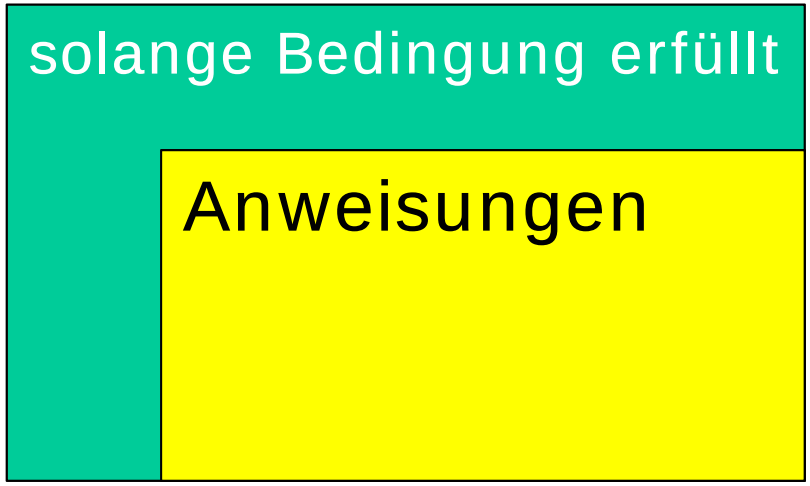
```
        do
                cout << „Endlosschleife“ << endl;
        while(1);
```

Struktogrammdarstellung

do...while-Schleife



while-Schleife



continue und break (1)

- Wenn man in der Blockanweisung einer Schleife vorzeitig an den Anfang (hier: zur while(…)-Prüfung) zurückkehren will,
  kann man die continue-Anweisung benutzen.
- Wenn man aus der Blockanweisung heraus die Schleife vorzeitig total verlassen (beenden) will, kann man die break-Anweisung benutzen.
- ist jedoch ähnlich verpönt wie die goto-Anweisungen…

```
while (Laenge < kurz) {
      Laenge++;
      if (Laenge%Sprung = 0) continue;
      if (Breite == Ziel) {
            cout << "Ziel getroffen" << endl;
            break;
      }
}
```

continue und break (2)

- So lange der C++-Ausdruck einer while-Schleife ***true*** ist, wird die Schleife nicht verlassen
- Setzt man die Startbedingung einer while-Schleife gleich **1**, erzeugt man eine sogenannte Endlosschleife
- Solch eine Schleife wird niemals verlassen, da mit der **1** die Bedingung immer ***true*** bleibt!
- Nur mit einem ***break*** kann eine solche Endlosschleife beendet bzw. verlassen werden:

```
while (1) {
      zaehler++;
      if (zaehler > 1000) break;
}
cout « "Zaehler = " « zaehler « "\n";
```

Bedingungs-Behandlung

Die Schleifenbedingung kann innerhalb der Schleife beliebig behandelt werden:

```
int bedingung = 0;
do {
      cout << "Hallo, guten Morgen! \n";
      cin >> bedingung;
}
while (bedingung > 0);
cout « "Auf Wiedersehen! \n";
```

Hier kann jetzt abhängig von der Eingabe *bedingung* so oft wie gewünscht ein „Hallo, guten Morgen!“ ausgegeben werden.

for-Schleife (1)

Mit einer for-Schleife kann man die drei Schritte
Initialisierung, Test und Inkrementierung zusammenfassen:

```
int bedingung;
for (bedingung = 0; bedingung < 5; bedingung++) {
//   Initialisierung;   Test;  Inkrementierung
      cout << "Hallo Schleife, guten Morgen!\n";
}
```

```
Reihenfolge:
1. Initialisierung (einmal zu Beginn der Schleife)
2. Prüfen, ob Bedingung erfüllt:
      - wenn ja (true) -> Anweisungsblock ausführen
      - wenn nein (false) -> Schleife verlassen
3. Inkrementierung und weiter bei Schritt 2.
```

for-Schleife (2)

Initialisierung, Test und Inkrementierung können auch mehrere oder komplexere Ausdrücke sein:

```
for (int k=0, m=9; k<3 || m<3; k++, m--)
      cout << "k = " << k << " / m = " << m << endl;
```

Ausgabe:

```
k = 0 / m = 9
k = 1 / m = 8
k = 2 / m = 7
```

Obacht:
Gut auf Komma und Semikolon unterscheiden!

for-Schleife (3)

Initialisierung, Test und Inkrementierung können auch leere Ausdrücke sein:

- Der Ausdruck for (; bedingung < 5;) ist eine while-Schleife:
  ~> ohne Behandlung der Bedingung im Block wird das eine Endlosschleife
- Die total leere for-Anweisung for (; ;) ist eine Endlosschleife!

Da der Ausdruck einer for-Schleife bereits die Ausführung mehrerer Anweisungen anbietet, kann die folgende Blockanweisung auch einmal leer sein:

```
for (k=0; k < 5; cout << "k= " << k++ << endl);
```

Geschachtelte for-Schleifen

- Die for-Schleife erlaubt eine beliebig tiefe Verschachtelung
- Das „Durchwandern“ von z.B. quadratischen Feldern A[z,s] wird ermöglicht durch eine Schleife-in-Schleife-Konstruktion

Ein Beispiel:

```
int z = 3 , s = 4;
for (int i = 0; i < z; i++) {
      for (int j = 0; j < s; j++)
            cout « " X ";
      cout « "\n";
}
```

Nach Beendigung der inneren Schleife wird die äußere Schleife um 1 erhöht und die innere Schleife beginnt von vorn.