# C++ Teil 3

## 3.3 Schleifen

Man kann bestimme Anweisungen in einem Programm mehrfach ausführen lassen. Dazu gibt es in C++ verschiedene Schleifen.

- `for`-Schleife
- `while`-Schleife
- `do-while`-Schleife

### for-Schleife

```
for ( Ausdruck1; Ausdruck2; Ausdruck3 )
      Anweisung
```

`Ausdruck1` ist die sogenannte **Initialisierung** und wird **vor** dem ersten Schleifendurchlauf ausgewertet. `Ausdruck2` ist die **Bedingung**, die für einen Schleifendurchlauf erfüllt sein muss. `Ausdruck3` wird am Ende der Schleife ausgeführt. In `Ausdruck1` können auch mehrere Variablen initialisiert werden, ebenso kann `Ausdruck3` mehrere Anweisungen enthalten. Es folgen zwei Beispielprogramme.

```
// 1) einfache for-Schleife
#include <iostream>
using namespace std;

int main()
{
      int zaehler;
      cout << "Schleife laeuft.\n\n";
      for (zaehler = 0; zaehler < 5; zaehler++)
            cout << "\nZaehler: " << zaehler << ".\n";
      return 0;
}
```

```
// 2) for-Schleife mit mehrfacher Initialisierung
#include <iostream>
using namespace std;

int main()
{
      for (int i=0, j=0; i<3; i++, j++)
            cout << "i: " << i << "  j: " << j << endl;
      return 0;
}
```

Schleifen können auch geschachtelt werden. Die **innere** Schleife wird für jeden Durchlauf der **äußeren** Schleife einmal komplett durchlaufen. Ein Beispiel: Nachdem der Benutzer eingegeben hat, wie viele Zeilen und Reihen auf den Bildschirm geschrieben werden sollen, läuft die äußere `for`-Schleife mit der Zählvariable `i` los. Die erste Zeile entsteht. Sie besteht aus der Anzahl der Reihen, die in der inneren Schleife mit der Zählvariablen `j` erzeugt werden. Nach Abarbeitung der inneren Schleife, wird die Zählvariable der äußeren Schleife um eins erhöht (`i++`) und die zweite Zeile entsteht, solange die Bedingung (`i<zeilen`) erfüllt ist; usw.

```
// 3) geschachtelte for-Schleifen
#include <iostream>
using namespace std;

int main()
{
      int reihen, zeilen;
      char buchstabe;
      cout << "Wie viele Zeilen? ";
      cin >> zeilen;
      cout << "Wie viele Reihen? ";
      cin >> reihen;
      cout << "Welches Zeichen? ";
      cin >> buchstabe;
      for (int i = 0; i<zeilen; i++)
      {
         for (int j = 0; j<reihen; j++)
            cout << buchstabe;
            cout << "\n";
      }
     return 0;
}
```

Es ist auch erlaubt, `Ausdruck1`, `Ausdruck2` oder `Ausdruck3` wegzulassen – nicht jedoch die Strichpunkte. Ist `Ausdruck2` leer, gilt er als wahr (`true`). Damit ist

```
for ( ; ; ) ...
```

eine Endlosschleife, die nur mit `break` oder `return` verlassen werden kann (siehe unten).

### while-Schleife

```
while (Ausdruck)
      Anweisung
```

Solange `Ausdruck` wahr ist, wird `Anweisung` ausgeführt. Der `Ausdruck` wird dabei am Anfang der Schleife, d.h. **vor** dem ersten Schleifendurchlauf, ausgewertet. Ist der `Ausdruck` also schon am Anfang nicht wahr (`true`), wird die Schleife kein einziges Mal durchlaufen.

```
// While-Schleife
#include <iostream>
using namespace std;

int main()
{
	int zaehler = 0;  // Initialisierung
	while(zaehler < 5) // Ist die Bedingung noch wahr (true)
	{
		zaehler++;  // Schleifenkörper
		cout << "Zaehler: " << zaehler << "\n";
	}
	cout << "Zaehlen beendet: " << zaehler << ".\n";
	return 0;
}
```

Vergleiche diese Programme mit dem entsprechenden im Abschnitt `for`-Schleife.

### do-while-Schleife

```
do
	Anweisung
while ( Ausdruck );
```

Die `do`-Bedingung wird **nach** jeder Ausführung der `Anweisung` ausgewertet. Daher wird die `Anweisung` im Gegensatz zur `for`- bzw. `while`-Schleife mindestens einmal ausgeführt.

```
// do-while-Schleife
// Ausgabe der kleinen Buchstaben
#include <iostream>
using namespace std;

int main()
{
	char ch = 'a';
	do
	{
		cout << ch << ' ';
		ch++;
		for (long int i=0; i<1000000; i++)
			; // verzögerte Ausgabe durch leere for-Schleife
	} while (ch <= 'z');
	return 0;
}
```

## 3.4 Sprünge

Es gibt vier Möglichkeiten, innerhalb einer Schleife den Ablauf zu beeinflussen.

- break (Abbruch)
- continue (erneuten Schleifendurchlauf starten)
- goto (Sprung)
- return (Verlassen einer Funktion)

Auf die Anweisung `goto` wird nicht eingegangen. Sie zerstört die Programmstruktur, erschwert die Lesbarkeit und Wartbarkeit eines Programms. Es entsteht der sogenannte **Spagetticode** (wie er z.B. in BASIC üblich ist). Auf die Anweisung `return` kommen wir in einem der nächsten Abschnitte zurück.

### break

Mit der `break`-Anweisung kann eine Schleife (oder ein `switch`-Block; siehe oben) verlassen werden.

```
// break-Anweisung
#include <iostream>
using namespace std;

int main()
{
     int i = 0, j;
     while (1)          // Endlosschleife
                        // Die Bedingung (1) ist immer wahr (true)
     {
        i++;
        if (i > 10)
            break;      // Mit break wird die Schleife verlassen
     }
     cout << "Zaehler: " << i;
     return 0;
}
```

### continue

Mit der `continue`-Anweisung wird an das Ende der Schleife gesprungen.

## Aufgaben

1. Was ist der Unterschied zwischen der `while`- und der `do-while`-Schleife?
2. Welchen Wert hat die Variable `x`, wenn folgende `for`-Schleife durchgelaufen ist?

   ```
   for (int x=0; x<100; x++)
   ```

3. Schreibe ein Programm, das mit Hilfe einer geschachtelten `for`-Schleife eine `10x10`-Matrix von Nullen (`0`) auf den Bildschirm schreibt.

4. Schreibe drei Programme, die von 100 bis 200 in 2er-Schritten zählen. Verwende dazu die

   a) `for`-Schleife.

   b) `while`-Schleife.

   c) `do-while`-Schleife.

5. Schreibe ein Programm, das in einer Tabelle (Tabulator) zu einem Wert in Grad Celsius ($C$) den zugehörigen Wert in Fahrenheit ($F$) ausgibt. Umrechnung: $F = C \cdot \frac{9}{5} + 32$ . Die Tabelle soll von –5 bis +15 °C gehen.

6. Die Fibonacci Zahlen (nach dem italienischen Mathematiker Leonardo von Pisa, nach 1240 genannt Fibonacci) lauten folgendermaßen: 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Die n-te Fibonacci Zahl ist die Summe aus der (n-1)-ten und der (n-2)-ten. Beispiele: die dritte Fibonacci Zahl ist die Summe aus der ersten (1) und der zweiten (1) Fibonacci Zahl, also 2. Die siebte Fibonacci Zahl ist die Summe aus der fünften (5) und der sechsten (8) Fibonacci Zahl, also 13.

   Schreibe ein Programm, das die ersten 100 Fibonacci Zahlen berechnet und auf dem Bildschirm ausgibt.