

# Schleifen

**Schleifen** sind **Anweisungen** und veranlassen den Programmfluss zur Wiederholung eines Teil des Codes. Dieser Teil wird **Schleifenrumpf** oder auch **Schleifenkörper** genannt und wird wiederholt, bis eine **Abbruchbedingung** eintritt. **Endlosschleifen** sind Schleifen, die eine Abbruchbedingung nie erreichen oder erst gar keine aufweisen.

- abweisende oder kopfgesteuerte Schleifen: `while` – Schleifen
- nichtabweisende oder fußgesteuerte Schleifen: `do-while` – Schleifen
- Zählschleifen (Sonderform von kopfgesteuerten Schleifen): `for` – Schleifen

## while – Schleife

- Syntax: 

```
while ( Ausdruck )
    Anweisung

while (log. Ausdruck)
{
    anweisung_1;
    ...
    anweisung_n;
};
```

solange log. Ausdruck wahr, führe aus...

- Beispielprogramm:

```
// Berechnung des Durchschnitts ganzer Zahlen
#include <iostream>
using namespace std;

main()
{
    int x, anzahl = 0;
    float summe = 0.0;

    cout << "Geben Sie eine ganze Zahl ein:\n"
         << "(Abbruch mit beliebigem Buchstaben)"
         << endl;
    while (cin >> x)
    {
        summe += x;
        ++anzahl;
    }
    cout << "Der Durchschnitt der Zahlen: "
         << summe / anzahl << endl;
}
```

Solange gültige Zahl eingelesen wird

Präfix-Increment: erhöhe anzahl um 1 vor Auswertung des zugehörigen Ausdrucks

## do-while – Schleife

- Syntax: 

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

```
do {
    anweisung_1;
    ...
    anweisung_n;
}
while ( Ausdruck );
```

Führe aus und wiederhole, wenn Ausdruck wahr
- abhängige Anweisungen werden in jedem Fall mind. einmal ausgeführt

## for – Schleife

- Syntax: 

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

```
for ( Ausdruck1; Ausdruck2; Ausdruck3 )
{
    anweisung_1;
    ...
    anweisung_n;
}
```

Ausdruck1 ... Initialisierung  
Ausdruck2 ... Laufzeitbedingung  
Ausdruck3 ... Reinitialisierung
- es wird genau einmal Ausdruck1 ausgeführt, dann wird Ausdruck2 zu Beginn jedes Schleifendurchlaufs bewertet:
  - wenn Ausdruck2 == false, so wird Schleife beendet
  - wenn Ausdruck2 == true, so werden Anweisungen im Rumpf ausgeführt  
-> anschließend wird Reinitialisierung vorgenommen mit Ausdruck3 und erneut Ausdruck2 geprüft
- Äquivalenz-Betrachtung:

```
int z = 1;
while ( z <= 10 )
{
    cout << z
        << ". Durchlauf\n";
    ++z;
}
```

```
int z;
for ( z=1; z<=10; ++z )
    cout << z
        << ". Durchlauf\n";
```

Variablendeklaration und Initialisierung hier möglich, siehe 3. Übung (if-Anweisung)

```
for ( int z=1; z<=10; ++z )
    cout << z
        << ". Durchlauf\n";
```

- da Anweisungen auch leer sein können, geht auch dies: `for ( ; ; ) ...`  
 -> Endlosschleife, da bei fehlendem Ausdruck2 die Laufzeitbedingung als "wahr" angenommen wird

`for ( ; Ausdruck; )` ← äquivalent → `while ( Ausdruck )`

## Schleifensprünge mit `break` und `continue`

### 1. `break` – Anweisung:

- führt zum unmittelbaren Verlassen der Schleife und es wird zur ersten Anweisung hinter dieser verzweigt

### 2. `continue` – Anweisung:

- bewirkt das Gegenteil von `break` – hier wird unmittelbar nächste Wiederholung begonnen
- bei `while`- und `do-while` – Schleife wird zur Laufzeitbedingung verzweigt, bei `for` – Schleife wird zur Reinitialisierung verzweigt

## Der Kommaoperator

Mit einem **Kommaoperator** können dort **mehrere Ausdrücke** angegeben werden, wo syntaktisch nur **ein Ausdruck** stehen darf.

- Syntax: `Ausdruck1, Ausdruck2 [, Ausdruck3 ... ]`

- Beispiel:

```
int x, i, grenze;
for ( i=0, grenze=8; i<grenze ; i+=2 )
    x = i * i, cout << setw(10) << x;
```

- Kommaoperator besitzt niedrigsten Vorrang aller Operatoren, demnach geht auch:

```
x = ( a = 3, b = 5, a * b )
```

← Erst Zuweisungen in Klammern ausführen, dann Wert von `a*b` an `x` übergeben

# Aufgabe "Messwerte"

## Aufgabenstellung

Es sollen über die Konsole positive Messwerte erfaßt und weiterverarbeitet werden. Die Eingabesequenz soll abgeschlossen werden, wenn ein negativer Messwert eingegeben wurde. Desweiteren sollen alle Messwerte, die  $< 10^{-3}$  und  $> 10^3$  sind ignoriert werden.

(a) Das Programm soll Anzahl aller gültigen Messwerte ermitteln und am Ende auf dem Bildschirm ausgeben.

(b) Von den gültigen Messwerten ist der Mittelwert zu bilden und am Ende des Programmlaufes auszugeben.

Beispielausgabe:

```
Messwert: 10.1
Messwert: 22.5
Messwert: 42.8
Messwert: 99.123
Messwert: 1500.34
Messwert: 58.77
Messwert: -1

Anzahl der Messwerte: 5
M i t t e l w e r t : 46.6586
```

## Analyse

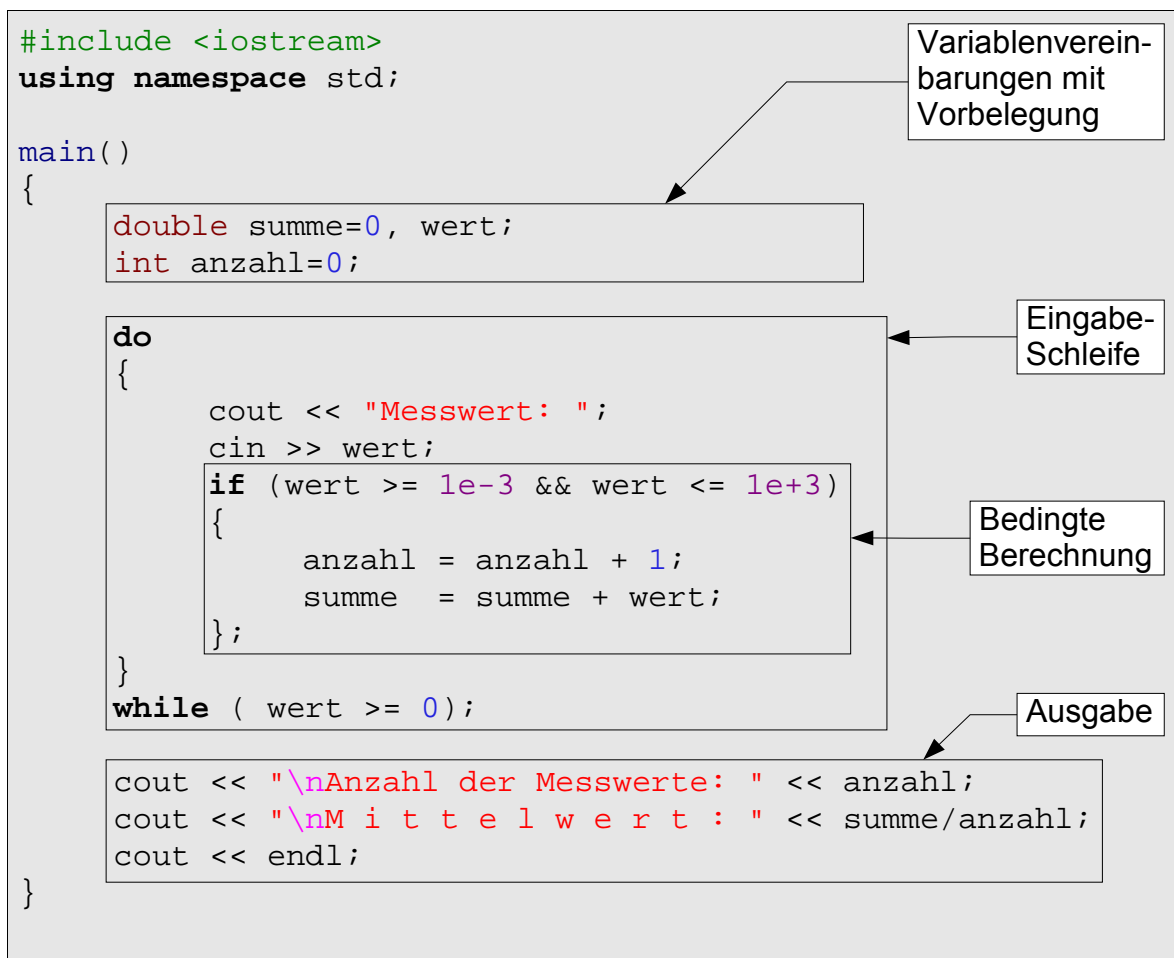
- benötigte Variablen:

<b>Bezeichner</b>	<b>Funktion</b>	<b>Datentyp</b>
wert	Messwert speichern	double
anzahl	Anzahl der gültigen Messwerte	int
summe	Summe über gültige Messwerte	double

- notwendige Schritte:
  1. Eingaben lesen
  2. Eingaben prüfen und Berechnungen durchführen
  3. Ausgabe des Ergebnisses
- zu beachten:
  - `anzahl` und `summe` sollten zu Beginn mit 0 initialisiert werden

- Pseudocode:
  1. Setze `anzahl` und `summe` auf 0
  2. Lies einen Messwert von der Eingabe in die Variable `wert`
  3. Ist `wert` eine gültige Eingabe, dann
    - erhöhe `anzahl` um 1 und `summe` um `wert`
  - sonst,
    - tue nichts
  4. wiederhole Schritt 1 und 2 solange `wert` positiv ist
  5. gib `anzahl` aus
  6. gib Mittelwert aus: `summe / anzahl`

## Quelltext-Erstellung



Frage: Was muß alles beachtet werden, wenn abweisende Schleife verwendet werden soll?