# Schleifen (Iteration)

## 1. Die while-Schleife

```
while (logischer ausdruck)
        anweisung
```

Die Anweisung wird solange ausgeführt, wie der logische Ausdruck wahr ist (Wert ≠ 0). Da der logische Ausdruck vor dem ersten Durchlaufen der Schleife ausgewertet wird, kann es sein, dass es gar nicht zur Ausführung der Anweisung kommt (**abweisende Schleife**).

## 2. Die do-while-Schleife

```
do
        anweisung
while (logischer ausdruck);
```

Die Anweisung wird solange ausgeführt bis der logische Ausdruck falsch ist (Wert = 0). Die Anweisung wird mindestens einmal ausgeführt (**nicht abweisende Schleife**).

## 3. Die for-Schleife

```
for (ausdruck_1; ausdruck_2; ausdruck_3)
        anweisung
```

Mit der for-Schleife hat man die Möglichkeit, einfache Zählvorgänge übersichtlich zu formulieren (**Zählschleife**). Ausdruck_1 initialisiert die Schleifenvariable, ausdruck_2 legt die Abbruchbedingung für die Schleife fest (es wird abgebrochen, wenn der ausdruck_2 falsch ist), und im ausdruck_3 wird die Schleifenvariable verändert (zumeist erhöht oder erniedrigt). In jedem Schleifendurchlauf wird die Anweisung ausgeführt.

**Anmerkung**: Jede for-Schleife kann durch die folgende while-Schleife ersetzt werden

```
ausdruck_1;
while (ausdruck_2)
        {
        anweisung
        ausdruck_3;
        }
```

### Anwendungen

- Wiederholbarkeit von Berechnungen

```
do
    {
    //Berechnungen
    cout << "Weitere Berechnungen? (J/N)";
    cin >>  antwort;
    }
while (toupper(antwort) ! = 'N');
```

**Hinweise**: Die Funktion **toupper** aus **<ctype.h>** wandelt Buchstaben in Großbuchstaben um. Die Variable **antwort** vom Typ **char** muss vereinbart werden.

**Aufgabe 1**:
Verbessern Sie das Programm zur Berechnung des Kugelvolumens durch Wiederholbarkeit der Berechnungen.

- Absicherung dafür, dass nur bestimmte Zeichen eingegeben werden können:
```
do
    cin >> antwort;
while (toupper(antwort) != 'N' && toupper(antwort) != 'J');
```

**Aufgabe 2:**
Arbeiten Sie in das Programm aus Aufgabe 1 zusätzlich noch obige Schleife ein.

**Aufgabe 3:**
Erstellen Sie ein Struktogramm und schreiben Sie ein Programm, das das Maximum von *n* nacheinander eingegebenen Zahlen ermittelt.

**Aufgabe 4:**
Was wird durch folgendes Programm auf dem Bildschirm ausgegeben?
```
#include <iostream.h>
int main()
{
        double x, y;
        double s = 0.1;
        for (x = 0; x < 1.1; x +=s)
        {
                y = x * x;
                cout << y << ", ";
        }
        return 0;
}
```

**Aufgabe 5:**
a) Erstellen Sie ein Struktogramm und schreiben Sie ein Programm, das folgenden Algorithmus zur Berechnung der so genannten Ulam-Folge realisiert: 1. Gib eine natürliche Zahl $u > 1$ ein. 2. Solange $u \neq 1$ ist, tue folgendes: Wenn $u$ gerade, dann ersetze $u$ durch $u / 2$, ansonsten ersetze $u$ durch $3 * u + 1$ und gib $u$ aus.
b) Testen Sie das Programm u. a. mit den Eingaben 95, 96, 97, 101, und 113. Vergleichen Sie die Ergebnisse und ziehen Sie Schlussfolgerungen daraus.
c) Ergänzen Sie das Programm durch Wiederholbarkeit der Berechnungen, Zählung der benötigten Schrittzahl bis zum Abbruch der Berechnungen, Ausgabe des größten der berechneten Folgeglieder.

**Aufgabe 6:**
Eine natürliche Zahl *n* heißt perfekt, wenn *n* als Summe aller natürlichen Zahlen *i* ($1 \leq i \leq n/2$), durch die *n* ohne Rest teilbar ist, dargestellt werden kann. Die Zahl 28 ist beispielsweise eine perfekte Zahl. Es ist $28 = 1 + 2 + 4 + 7 + 14$ und 1, 2, 4, 7, 14 sind genau alle Zahlen, durch die 28 ohne Rest teilbar ist. Schreiben Sie ein Programm, das alle perfekten Zahlen zwischen 1 und 1000 auf dem Bildschirm ausgibt.