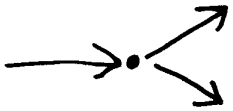
2.4.6. Kontrollstrukturen

if-Anweisung: Bedingte Ausführung (Verzweigung)



2 Varianten:

`if` (Bedingung) Anweisung

(Anweisung = einzelne Anweisung oder Block)
Bedeutung: die Anweisung wird nur ausgeführt, wenn die Bedingung den Wahrheitswert true (also einen Wert $\neq 0$) liefert
sonst wird sie übersprungen

```
if (Bedingung)
    Anweisung1
else
    Anweisung2
```

Bedeutung: wenn die Bedingung true liefert, wird Anweisung1 ausgeführt, sonst Anweisung2

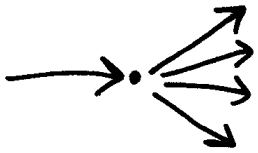
Schachtelung der 2. Variante (Bedingungs-Kaskade):

```
if (B1) A1
else
    if (B2) A2
    else
        if (B3) A3
        else
            if (B4) A4
            else A
```

(kann leicht unübersichtlich werden)

alternative Konstruktion:

switch-Anweisung (Mehrfach-Verzweigung)



```
switch (Ausdruck)
    {
    case Konst1 :  Anweisungen
                   break;
    case Konst2 :  Anweisungen
                   break;
    ....
    case Konstn :  Anweisungen
                   break;
  [ default :      Anweisungen ]
    }
```

Bedeutung:
1. Ausdruck wird ausgewertet
2. es wird hinter demjenigen **`case`** begonnen, wo die Konstante mit dem Wert übereinstimmt.
   Wenn **`default`** auftritt, wird dies zum Einstieg benutzt, falls der Wert mit keiner der Konstanten übereinstimmt.
3. **`break;`** führt zum Verlassen von *switch*.

Iteration:
wiederholte Ausführung gleicher Rechenanweisungen mit verschiedenen Werten.

"Schleife"

wichtig: bei jedem Schleifendurchlauf muss eine *Abbruch-bedingung* überprüft werden (bzw. *Schleifenbedingung* = Negation der Abbruchbedingung).

in C: 4 Schleifentypen (*while*, *do ... while*, *for*, *goto*).

`while` (Schleifenbedingung) Anweisung

1. Schleifenbedingung wird geprüft
2. falls erfüllt (Wert ≠ 0): Anweisung wird ausgeführt (ggf. Block), weiter bei 1.
3. falls nicht erfüllt (Wert == 0): *while*-Konstrukt wird verlassen.

`do` Anweisung **`while`** (Schleifenbedingung);

entspricht
Anweisung **`while`** (Schleifenbedingung) Anweisung
d.h. die Anweisung muss in jedem Fall mindestens 1mal ausgeführt werden.

`for` (Initialisierung; Schleifenbedingung; Anw1) Anw2

entspricht

```
Initialisierung
while (Schleifenbedingung)
    { Anw2 Anw1; }
```

Anw1 enthält in der Regel eine Inkrementierung (oder Dekrementierung) einer Laufvariablen

"Außerplanmäßiger" Abbruch einer Schleife:

`break;`

- nur innerhalb von Schleifen (while, do, for) oder switch
- bewirkt das sofortige Verlassen
- bei geschachtelten Schleifen bzw. switches: **`break`** terminiert die innerste

`continue;`

- nur innerhalb von while-, do- oder for-Schleifen
- Abbruch nur eines (des aktuellen) Schleifendurchlaufs
- bei while oder do: Steuerung wird an die Schleifenbedingung übergeben, bei for: an die Inkrementierung (Anw1).
- ggf. erfolgen weitere Schleifendurchläufe.

Abbruch des ganzen Programms:

`exit`(Fehlercode)**`;`**

Der Fehlercode (*int*) signalisiert eventuelle Fehler an aufrufende Prozesse
Fehlercode == 0 steht für fehlerfreie Ausführung.
Abbruch aufgrund v. Fehler also z.B.

```
exit(1);
```

Tip: vorher Text ausgeben, der den Fehler beschreibt!

in **`main`** ist **`return n;`** aquivalent zu **`exit(n);`** .

Zur Fehlerbehandlung:

In C werden oft Fehler, die in Standardsituationen auftreten, nicht automatisch abgefangen.

Beispiele:

- Öffnen einer Datei (Datei existiert nicht, oder ist lesegeschützt...)
- Allozieren von dynamischem Speicherplatz (Speicher ist voll...)
- Bereichsüberschreitung beim Index eines Arrays

⇒ man muss selber für entsprechende Abfragen sorgen, um undefinierten "Absturz" des Programms auszuschließen!

Beispiel (hier Vorgriff auf Arbeiten mit Dateien):
Öffnen einer Datei zum Lesen

```
FILE *datei;
char c;
char dname[20] = "xxx.dat";
datei = fopen(dname, "r");
```

dies würde bei korrekter Ausführung zum Öffnen der Datei reichen, ist aber unsicher.
Deshalb anstatt der letzten Zeile:

```
if ((datei = fopen(dname, "r")) == NULL)
    {
    printf("\nDatei konnte nicht geöffnet"
         " werden!  Programmabbruch, bitte"
         " <return> druecken:\n");
    c = getchar();
    exit(1);
    }
```

Rückgabewert NULL bei **`fopen`** (und damit bei der Zuweisung) zeigt Misserfolg an.

In JAVA *erzwingt* der Compiler in solchen Fällen, dass eine *exception* (Ausnahmebedingung, Fehler) mit einer **`catch`**-Anweisung "aufgefangen" wird:

```
try
    {
    FileReader datei = new FileReader("xxx.dat");
    ...
    Anweisungen
    ...
    }  catch (IOException e)
        {
        System.out.println("\nDatei konnte nicht
            geöffnet werden" + e.toString());
        }
...
```

Labels (Marken):

- gleiche Syntax wie Variablennamen (Bezeichner), gefolgt von einem Doppelpunkt
- markieren eine Stelle innerhalb einer Funktion

goto-Befehl (Vorsicht! sparsam verwenden):

bewirkt Sprung zu einem Label
(geht nur innerhalb derselben Funktion)

- damit auch Sprung aus Schleifen heraus möglich

kann u.U. zur Fehlerbehandlung sinnvoll sein – Beisp.:

```
    for (...)
        for (...)
            {
            ...
            if (Katastrophe)
                goto error_marke;
            ...
    ...
    error_marke:
        Überreste abräumen
    ...
```

Sprünge mit **`goto`** und Labels:

- führen schnell zu unübersichtlichem Code
- lassen sich stets auch durch andere, übersichtlichere Kontrollstrukturen (if, while...) ersetzen

2.4.7. Grundsätze guter Programmcode-Gestaltung

- Programm übersichtlich schreiben (Einrückungen, Leerzeichen, Abstände...)
- Kommentare verwenden **`/* C-Kommentar */`**
- "sprechende" Variablen- und Funktionsnamen verwenden
- Umlaute und "ß" vermeiden
- Ökonomie/Eleganz: Doppelte Berechnungen vermeiden
- Modularisierung: Auslagerung von Berechnungen in Funktionen ($\rightarrow$ Übersichtlichkeit, Wiederverwendbarkeit)
- Verwendung globaler Variablen auf das Nötigste beschränken
- literale Konstanten (numerische Werte) im Code vermeiden, stattdessen Konstanten deklarieren
- Reihenfolge der Berechnungen stets so wählen, dass Zwischenwerte nicht zu groß oder zu klein werden
- alle Ausnahmesituationen abfangen (z.B. Nenner fast = 0; Öffnen von Datei schlägt fehl, Bereichsüberschreitungen...)