Die `if`-Anweisung

Syntax:
```
if (Bedingung)        // die runden Klammern sind Pflicht!
     genau-eine-Anweisung;
```

oder

```
if (Bedingung)
{    Eine-Anweisung;
     Noch-eine-Anweisung;
     ...
}
```

Die abhängige Anweisung bzw. der Anweisungsblock wird nur ausgeführt, wenn die Bedingung wahr ist.

Die Bedingung ist in C (also auch in C++ gestattet) irgendein numerischer Ausdruck. Ein Wert von 0 gilt als „falsch“ und ein Wert ungleich 0 als „wahr“. In C++ kann **(bzw. sollte!)** auch ein Ausdruck mit dem Ergebnistyp **`bool`** verwendet werden (in Java **muss** ein Ausdruck mit Ergebnis boolean verwendet werden)**.** Einrücken zur optischen Verdeutlichung der Abhängigkeit gehört zum guten Stil und wird dringend empfohlen. Ich schlage die Einrückung um eine Tabulatorposition (= 4-8 Zeichen) vor, damit die Abhängigkeit klar ersichtlich ist. Kernighan/Ritchie würden die öffnende Klammer auf derselben Zeile wie die Bedingung schreiben.

Beispiele:

```
if (1)     // ok in C/C++, falsch in Java (kein boolean)
     cout << "diese Anweisung wird immer ausgeführt\n";

if (true) // C++ und Java
     cout << "diese Anweisung wird immer ausgeführt\n";

if (x > 7)
     cout << "x ist zu gross\n";

if (x > 7)      // ab 2 Anweisungen MUSS man Klammern { } verwenden
{    cout << "x ist zu gross und wird durch 5 ersetzt\n";
     x = 5;
}

if (x > 7) {    // K&R Klammernsetzung , ist nur optisch unterschiedlich
     cout << "x ist zu gross und wird durch 5 ersetzt\n";
```

```
        x = 5;
}
```

Fehler:

```
if (i > 5)
     cout << "i ist zu gross\n";
     return EXIT_SUCCESS;      // diese Anweisung steht NICHT im  if

if (i > 5);                    // der Klassiker:  falscher  Strichpunkt
     cout << "i zu gross\n";   // diese Anweisung steht NICHT im if
```

Der Fehler mit dem falschen Strichpunkt wird **nicht vom Compiler erkannt**, da es sich hierbei um keinen Syntaxfehler handelt. Solche Patzer sind extrem schwer zu finden, da optisch fast alles richtig aussieht. Neuere Compiler warnen bei Option `-Wall`.

Fehler:

```
if (i = 0)      // Achtung: das ist eine Zuweisung, sollte sein :   i == 0
     cout << "i darf nicht 0 sein!\n";
```

Solche Fehler passieren in C/C++ oft, weil die Zuweisung (ein Gleichheitszeichen) und die Abfrage auf Gleichheit (logischer Operator ==) so ähnlich sind. Dummerweise ist diese Konstruktion in C/C++ erlaubt, auch wenn sie meistens völlig sinnfrei ist. Der Compiler gibt deshalb nicht einmal eine Warnung aus (der Gnu-Compiler warnt immerhin bei der Option `-Wall` für alle Warnungen). In Java wäre das fast immer ein Syntaxfehler, da das Ergebnis der Zuweisung meistens kein Wahrheitswert ist.

Die `else` Anweisung

Die `if`-Anweisung **kann** mit einer `else`-Anweisung kombiniert werden (muss es aber nicht). In diesem Fall **muss** die `else`-Anweisung **direkt auf die `if`-Anweisung folgen**. Genauso wie beim `if` kann man genau eine Anweisung oder aber einen Anweisungsblock zwischen geschweiften Klammer { } verwenden. Diese Anweisung(en) kommen genau dann zur Ausführung, wenn die `if`-Bedingung nicht zutrifft (Alternative). Auch hier empfehle ich das Einrücken.

Beispiel:

```
if (x > 7)
     cout << "x ist zu gross\n";
```

```
else
     cout << "x ist gerade richtig\n";
```

Fehler:

```
if (x > 7);     // dieser Strichpunkt ist die leere Anweisung
     cout << "x ist zu gross\n"; // NICHT im if
else       // Syntaxfehler: else folgt nicht direkt auf das if
     cout << "x ist gerade richtig\n";
```

Die `else if` Anweisung

Lässt man vom `else` wieder eine `if`-Anweisung abhängen, so spricht man von einer `else if` Konstruktion. Eine solche gibt es in C/C++ eigentlich nicht wirklich, sondern es ist eben eine Kombination von `else` mit einem weiteren `if`.

Man rückt aber anders ein, um Platz zu sparen:

schlecht:

```
if (x > 7)
     cout << "x ist zu gross\n";
else
     if (x < -5)
          cout << "x ist zu klein\n";    // zu weit rechts
```

besser:

```
if (x > 7)
     cout "x ist zu gross\n";
else if (x < -5)
     cout << "x ist zu klein\n";
```

Dadurch kann man wirklich von einem `if - else if - else` Block sprechen :

```
if (x < -70)
     cout << "x ist viel zu klein\n";
else if (x < -7)
     cout << "x ist zu klein\n";
else if (x > 7)
     cout << "x ist zu gross\n";
else
     cout << "x ist gerade richtig\n";
```

Vergleiche obigen Code mit der äquivalenten Variante ohne `else if`

```
if (x < -70)
     cout << "x ist viel zu klein\n";
else {
     if (x < -7)
          cout << "x ist zu klein\n";
     else {
          if (x > 7)
               cout << "x ist zu gross\n";
          else
               cout << "x ist gerade richtig\n";
     }
}
```

Die Bedingungen werden der Reihe nach überprüft. Die **erste zutreffende Bedingung** bestimmt die auszuführende Anweisung (oder den ganzen Anweisungsblock in `{}`) und beendet den ganzen `if`-Block. Trifft keine der Bedingungen zu und ist ein reines `else` vorhanden, so wird dessen Anweisung ausgeführt.

Da die Bedingungen der Reihe nach überprüft werden, sollte man die häufiger zutreffenden Bedingungen am Anfang anordnen.

Kommen mehrere `if`-Anweisungen vor und nur ein `else`, so ist es manchmal schwierig zu erkennen, wohin letzteres gehört. **In diesem Fall sollte man immer klammern:**

```
if (x > 7)
     if (x > 70)
          cout << x ist viel zu gross\n";
     else      // nicht (x > 70)
          cout << "x ist zu gross\n";
```

genau dasselbe, aber optisch eindeutig:

```
if (x > 7) {
     if (x > 70)
          cout << "x ist viel zu gross\n";
     else
          cout << "x ist zu gross\n";
}
```

die andere Variante (die etwas ganz anderes bewirkt): hier **muss** man Klammern setzen:

```
if (x > 7) {
     if (x > 70)
          cout << "x ist viel zu gross\n";
} else          // nicht (x > 7)
     cout << "x ist ok\n";
```

Ich möchte noch einmal betonen, dass das korrekte Einrücken nur für den Programmierer wichtig ist. Der Compiler ignoriert alle Arten von Einrückungen bei der Übersetzung.

if und C++17: if mit Init

Seit C++17 ist eine weitere Form des `if`-Statements erlaubt: `if` mit `Init`

```
if (Init; Bedingung)
     genau-eine-Anweisung;
```

`Init` ist meist eine Variablendefinition. Solche Variablen stehen im gesamten `if`-Block und auch (falls vorhanden) im `else`-Block zur Verfügung, sind aber außerhalb des `if` nicht mehr definiert:

```
if (double d{b*b - 4*a*c}; d > 0)        // d (Diskriminante) positiv
     // 2 reelle Loesungen
else if (d == 0)                         // hier existiert d noch
     // reelle Doppel-Loesung
else                                     // Diskriminante negativ
     // 2 konjugiert-komplexe Loesungen

// ab hier existiert d nicht mehr
```