1.5.F Programmstrukturen/Verzweigung – Ergänzungen und Bilder

1.5.F.1 Programmblöcke

Bei der Verzweigung in C wird der if-Zweig im Quelltext meistens durch ein Paar geschweifter Klammern umfasst. Er beginnt mit dem Zeichen `{` und endet mit dem Zeichen `}`. Einen solchen Bereich zwischen diesen beiden Zeichen nennt man *Verbundanweisung* oder auch *Block*:

```
1 if (x==3)
2 {    /* Anfang des Blockes */
3 }    /* Ende des Blockes */
4 else
5 {    /* Anfang des Blockes */
6 }    /* Ende des Blockes */
```

Ein Block ist damit wie ein Behälter für Anweisungen. Der Behälter kann leer sein, er kann aber auch eine oder mehrere Anweisungen enthalten.

Ein weiteres Beispiel für einen Block finden wir in *jedem* Programm: Der Rumpf des Funktionsbausteins `main()` ist ebenfalls ein solcher Block, auch er beginnt mit `{` und endet mit `}`:

```
1 int main(void)
2 {    /* Anfang des Blockes */
3 }    /* Ende des Blockes */
```

Ein Block darf aber auch im Quelltext stehen, ohne dass er zu einer Verzweigung oder zu einem Funktionsbaustein gehört:

```
1  #include <stdio.h>
2  int main(void)
3  {
4      printf("Vor dem Block - ");
5      {                                /* Anfang des Blockes */
6          printf("Im Block - ");
7          printf("Immer noch im Block - ");
8      }                                /* Ende des Blockes */
9      printf("Nach dem Block");
10     return 0;
11 }
```

Zwischen den Zeilen 5 und 8 steht ein Block. Dieser Block enthält die Anweisungen in den Zeilen 6 und 7. Die Blockklammern bewirken hier nichts Besonderes. Lässt man die Zeilen 5 und 8 weg, funktioniert das Programmstück genauso wie vorher.

1.5.F.2 Block als Verbundanweisung

Die einzige Aufgabe eines Blockes ist es also, dass er Anweisungen zusammenfasst. Zu diesem Zweck wirkt der Block im Quelltext nach außen wie eine einzige Anweisung (darum nennt man den Block ja auch Verbundanweisung).

Aus diesem Grund kann man an im if- und im else-Zweig (und ebenso nach `while`, `do` und `switch`) die Blockklammern weglassen, **wenn** der Zweig nur *eine einzige Anweisung* enthält:

```
1     if(x==3)
2     {
3         printf("x ist drei.\n");
4     }
5     else
6     {
```

```
7       printf("x ist gar nicht drei.\n");
8   }
```

Dieses Beispiel kann man kürzer schreiben, wenn man unbedingt will:

```
1   if(x==3)
2       printf("x ist drei.\n");
3   else
4       printf("x ist gar nicht drei.\n");
```

Im oberen Beispiel besteht der if-Zweig aus einem Block, also *einer* Verbundanweisung. Im unteren Beispiel besteht der if-Zweig aus *einer* gewöhnlichen Anweisung. Hat man in einem Zweig mehrere Anweisungen, darf man die Blockklammern nicht weglassen!

1.5.F.3 Vereinbarung von Variablen im Block

Falls man eine Variable nur innerhalb eines Blockes braucht, kann man sie zu Beginn des Blockes vereinbaren. Außerhalb ist sie dann nicht benutzbar. Man sagt, sie ist nur innerhalb des Blockes *sichtbar*:

```
1   int x;
2   scanf("%i", &x);
3   {
4       int y;
5       y=x*x*x;
6       printf("x hoch 3=%i\n", y);
7   }
```

In diesem Beispiel ist y nur von Zeile 4 bis 6 sichtbar. Es gilt als guter Stil, Variablen, die nur in bestimmten Bereichen gebraucht werden, zum Blockbeginn zu vereinbaren. Dazu später mehr.

1.5.F.4 Geschachtelte Blöcke

Ein Block ist also ein Behälter für Anweisungen. Und das Gute ist: Man kann in einen Behälter einen anderen Behälter hineinstellen. So, wie man einen kleinen Topf in einen großen Topf stellen kann oder eine kleine Schachtel in eine große Schachtel. Dieses Prinzip heißt somit *Schachtelung* von Blöcken: Ein großer Block kann einen oder mehrere kleine Blöcke enthalten und die wiederum noch kleinere Blöcke:

```
1   printf("Hier sind wir vor Block A.\n");
2   {
3       printf("Block A (aussen) hat begonnen.\n");
4       {
5           printf("Block B (innen) ist hier.\n");
6       }
7       printf("Block A (aussen) laeuft noch immer.\n");
8       {
9           printf("Block C (innen) ist hier.\n");
10          {
11              printf("Block D (ganz innen).\n");
12          }
13      }
14      printf("Block A (aussen) wird gleich enden.\n");
15  }
16  printf("Hier sind wir hinter Block A.\n");
```

1.5.F.5 Geschachtelte Blöcke und Verzweigung

In der Praxis wird die Schachtelung oft benutzt, um mehrere Verzweigungen miteinander zu kombinieren:

```
1    if(x>7)
2    {
3        printf("In␣diesem␣Block␣ist␣x␣groesser␣als␣7.\n");
4        if(x<11)
5        {
6            printf("In␣diesem␣Block␣ist␣x␣>␣7␣und␣x␣<␣11.\n");
7        }
8    }
```

Hier kann der innere Block nur dann betreten werden, wenn vorher der äußere Block betreten wurde. Auf diese Art wird eine UND-Verknüpfung realisiert: Der innere Block wird nur betreten, wenn `x>7` UND `x<11` ist.

1.5.F.6 Konventionen der Programmierer

In C kann man den Quelltext optisch so aufbereiten, wie man es will[1]. Es ist möglich, ein C-Programm in eine sehr, sehr lange Zeile zu packen (nur die include-Anweisungen müssen auf einer eigenen Zeile stehen). Genauso könnte man ein Programm auch Wort für Wort untereinander schreiben — dem Compiler ist das egal. Er orientiert sich an Klammern, Semikolons usw. Aber gut lesbar für Menschen ist das nicht. Es gibt daher einige Gebräuche (=Sitten, Übereinkommen), mit denen Programmierer einander helfen, Programme besser zu verstehen. Diese Gebräuche heißen *Konventionen*.

1.5.F.7 Konventionen für die Formatierung von C-Quelltexten

Hier sind einige dieser Konventionen für die optische Aufbereitung (die so genannte *Formatierung*) von Quelltexten in der Sprache C:

- Variablennamen bestehen aus Kleinbuchstaben.
- Konstantennamen bestehen aus Großbuchstaben.
- Die Namen von Variablen und Konstanten sind so gewählt, dass man ihren Zweck daraus erahnen kann.
- In jede Zeile wird eine Anweisung geschrieben. Ist die Zeile zu lang (mehr als 60–70 Zeichen), wird sie in der folgenden Zeile fortgesetzt.
- Komplizierte oder undurchsichtige Anweisungen werden am Ende der Zeile(n) durch Kommentare erläutert.
- Blöcke werden eingerückt (siehe unten).

1.5.F.8 Einrückung in C-Quelltexten

Was ist nun mit *Einrückung* gemeint? Zunächst meint Einrückung, dass eine oder mehrere Zeilen nicht ganz links am Rand der Seite beginnen, sondern ein paar Zeichen weiter rechts:

```
1 Diese Zeile beginnt ganz links.
2    Und diese Zeile ist um drei Zeichen eingerueckt.
```

[1] Nur den Leerraum zwischen Worten darf man nicht weglassen.

In C es üblich (=Konvention), dass man so eine Einrückung benutzt, um einen Block zu kennzeichnen. Das geht so:

a) **Nach** jeder Zeile, die mit einem `{` beginnt, werden am Zeilenanfang drei Leerzeichen mehr als bisher eingefügt.

b) **Ab** jeder Zeile, die mit einem `}` beginnt, werden am Zeilenanfang drei Leerzeichen weniger als bisher eingefügt.

Hier nun ein Beispiel:

```
1 printf("EINS\n");
2 if(x==1234)
3 {
4     printf("ZWEI\n");
5     printf("DREI\n");
6 }
```

Nach Zeile 3 wird also eingerückt (Blockanfang), und ab Zeile 6 wird die Einrückung wieder beendet (Blockende). Die geschweiften Klammern selbst sind also nicht von der Einrückung betroffen. Das ist sinnvoll: Die geschweiften Klammern sind auf derselben Spalte (also übereinander) wie das Wort `if`. Man sieht auf den ersten Blick, dass der if-Zweig von Zeile 3 bis Zeile 6 reicht[2].

Bei verschachtelten Blöcken merkt man die Vorteile solcher Konvention ganz besonders:

```
1  printf("EINS\n");
2  if(x==1234)
3  {
4      printf("ZWEI\n");
5      if(y==x-1)
6      {
7          printf("ZWEIEINHALB\n");
8          printf("ZWEIDREIVIERTEL\n");
9      }
10     else
11     {
12         printf("ZEISIEBENACHTEL\n");
13     }
14     printf("DREI\n");
15 }
```

Hier haben wir im äußeren Block zwei innere Blöcke, nämlich den if- und den else-Zweig einer Verzweigung. Man sieht genau, wo die Verzweigung beginnt (Zeile 5) und wo sie endet (Zeile 13).

Graphisch dargestellt wird diese Tatsache (äußerer Block enthält zwei innere Blöcke) in Abbildung 1. Der äußere Block ist schwarz dargestellt, die inneren rot und grün.

Wenn man einen Quelltext untersucht, ist es durchaus sinnvoll, die Blöcke von Hand zu markieren (es sei denn, man hat einen Editor, der das macht). Abbildung 2 zeigt, wie so eine Markierung aussehen kann. Dabei werden die Schlüsselwörter `if` und `else` markiert und mit den entsprechenden Blockgrenzen verbunden.

[2]Manche C-Programmierer benutzen andere Konventionen, die diesen Vorteil nicht ausnutzen. Hier wird die Konvention eingeführt, die für Anfänger am besten zu lernen ist.

```
printf("ZWEI\n");
if(x==1234)
{
     printf("ZWEI\n");
     if(y==x-1)
     {
          printf("ZWEIEINHALB\n");
          printf("ZWEIDREIVIERTEL\n");
     }
     else
     {
          printf("ZWEISIEBENACHTEL\n");
     }
     printf("DREI\n");
}
```

Abbildung 1: Verschachtelte Blöcke und Einrückung: Graphische Darstellung

```
printf("ZWEI\n");
if (x==1234)
{
     printf("ZWEI\n");
     if (y==x-1)
     {
          printf("ZWEIEINHALB\n");
          printf("ZWEIDREIVIERTEL\n");
     }
     else
     {
          printf("ZWEISIEBENACHTEL\n");
     }
     printf("DREI\n");
}
```

Abbildung 2: Verschachtelte Blöcke und Einrückung: Markieren des Quelltextes

1.5.F.9 Ersatz von UND durch Programmstruktur

a) Autofahren ohne Begleitung ist erlaubt mit Fahrerlaubnis ab Vollendung des 18 Lebensjahres

b) Das heißt: Alter ab 18 UND Führerschein → man darf alleine fahren

c) Code:

```c
#include <stdio.h>
int main(void)
{
    int alter=19;
    int fahrerlaubnis=1;
    if(alter>=18)
    {
        if(fahrerlaubnis)
        {
            printf("Sie duerfen allein Auto fahren!\n");
        }
    }
    return 0;
}
```

1.5.F.10 Ersatz von ODER durch Programmstruktur

a) Die Teilnahme an einer Sylvesterfeier kostet 60EUR. Das Büffet mit Essensauswahl und Getränken kostet einmal zusätzlich 25EUR.

b) Das heißt: Essen gewünscht ODERAUCH Trinken gewünscht → Kosten+=25EUR.

c) Code:

```c
#include <stdio.h>
int main(void)
{
    int essen=1;
    int trinken=1;
    double kosten=60.00;
    if(essen)
    {
        kosten += 25.00;
    }
    else
    {
        if(trinken)
        {
            kosten += 25.00;
        }
    }
    return 0;
}
```