1.7 Programmstrukturen/Zählschleife

1.7.1 Vereinfachung eines Sonderfalls

Die Programmierung von Schleifen kommt häufig vor, ist aber nicht immer ganz einfach und birgt Fehlerquellen wie Endlosschleifen und das Zaunpfahlproblem (Schleife einmal zu selten oder einmal zu oft durchlaufen).

Viele Schleifen folgen aber einem einfachen Grundmuster. Ein Beispiel zeigt `quadratzahl1.c`:

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int lauf;
6
7      lauf=3;
8      while(lauf<8)
9      {
10         printf("Quadratzahl: %d\n", lauf*lauf);
11         ++lauf;
12     }
13     return 0;
14 }
```

In diesem Programm wird eine Variable von 3 bis 7 hochgezählt. Dabei wird das Quadrat der jeweiligen Zahl ausgegeben.

Diese Konstruktion (Hochzählen einer Variable – Benutzen der Variable) wird tatsächlich in der Informatik häufig gebraucht. Es gibt dafür den Namen **Zählschleife**. Bei einer Zählschleife ist die Anzahl der Durchläufe vorher bekannt.

1.7.2 Schleifenkonstruktion mit `for`

Im obigen Beispiel wurde die Zählschleife mit einer `while()`-Anweisung erstellt. `while()` ist jedoch für die kopfgesteuerte Schleife gedacht. Gesucht ist nun eine passende Realisierung der Zählschleife in C. In vielen Programmiersprachen sind Zählschleifen realisiert. Meistens werden sie dort mit dem Schlüsselwort `for` eingeleitet. Auch in C gibt es eine Schleife, die mit `for` beginnt. Das Quadratzahl-Programm sieht damit so aus (`quadratzahl2.c`):

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      unsigned int lauf;
6
7      for(lauf=3; lauf<8; ++lauf)
8      {
9          printf("Quadratzahl: %d\n", lauf*lauf);
10     }
11     return 0;
12 }
```

In der Klammer der Zeile 7 stehen drei Ausdrücke:

- `lauf=3`: Vor Beginn der Schleife wird dies einmal ausgeführt. Mit diesem Ausdruck wird die Schleife initialisiert.

- `lauf<8`: Dies ist die Eintrittsbedingung. Wenn die Bedingung erfüllt ist, wird der aktuelle Durchlauf ausgeführt, ansonsten wird hinter die Schleife gesprungen.
- `++lauf`: Dies ist die Weiterschaltanweisung. Sie wird am Ende des Rumpfes bei jedem Durchlauf einmal ausgeführt.

Es zeigt sich: Die beiden obigen Quelltexte wirken vermutlich gleich. Nun soll ausprobiert werden, ob das auch für die compilierten Programme stimmt:

Terminal

```
schueler@debian964:~$ gcc -o quadratzahl1 quadratzahl1.c
schueler@debian964:~$ gcc -o quadratzahl2 quadratzahl2.c
schueler@debian964:~$ strip quadratzahl1 quadratzahl2
schueler@debian964:~$ diff -s quadratzahl1 quadratzahl2
Dateien quadratzahl1 und quadratzahl2 sind identisch.
```

Offenbar ist die `for()`-Anweisung *in C* nur eine andere Schreibweise für eine kopfgesteuerte Schleife. **In C gibt es keine echte Zählschleife.** Der Ausdruck `for(a;b;c){x;}` ist in C nur eine andere Schreibweise für `a;while(b){x;c;}` (siehe Abbildung 1).

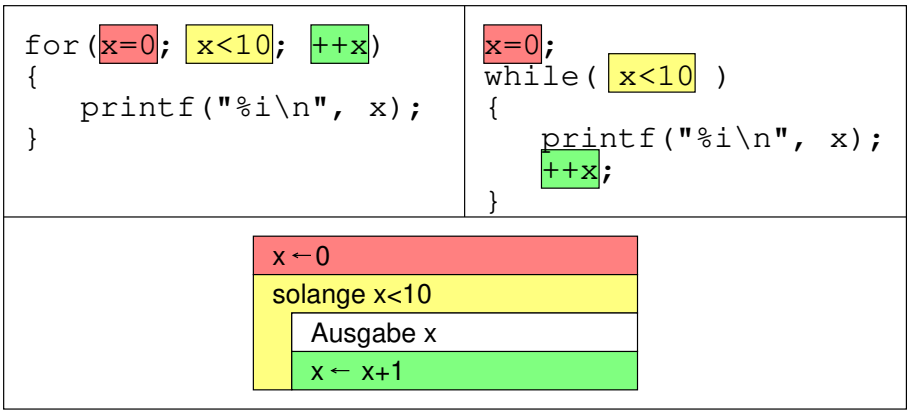

Abbildung 1: Struktogramm zur for-Anweisung in C

1.7.3 Bedienungsanleitung für `for`

Wenn die `for`-Anweisung schon keine echte Zählschleife ist, muss man durch eigene Programmierregeln die Benutzung einschränken:

- Man sollte `for` wirklich nur für Zählschleifen benutzen. Jeder der drei Ausdrücke in der `for`-Klammer kann nämlich beliebige Ausdrücke enthalten. Man kann auch jeden der Ausdrücke weglassen, sogar alle drei (man erhält mit `for(;;){}` eine Endlosschleife). Alle diese Möglichkeiten nützen nichts, sorgen aber für schwer lesbare Programme.
- Die Zähl- oder Laufvariable sollte ganzzahlig sein, wenn möglich vorzeichenlos. Damit werden Endlosschleifen vermieden.
- Die Schleife sollte nie bis ans Ende des Zahlenbereichs laufen, da sonst die Schleifenbedingung immer wahr ist und man wieder eine Endlosschleife erhält. An dieser Stelle merkt man, dass man es nicht mit einer echten Zählschleife zu tun hat.
- Die Zähl- oder Laufvariable kann in der Schleife verändert werden. Das sollte man unbedingt vermeiden, um keine schwer zu findenden Fehler zu erhalten.
- Es empfiehlt sich, in der Schleifenbedingung nicht auf == oder `!=`, sondern auf < oder > zu testen. Damit endet die Schleife auch dann, wenn der Zielwert verfehlt wurde.