1.6 Programmstrukturen/Abweisende Schleife

1.6.1 Schleife

Die meisten Programmiersprachen haben Konstruktionen, die eine beliebige Wiederholung von Programmteilen ermöglichen. Diese Konstruktionen heißen Schleifen. Eine dieser Schleifen in C ist die while-Schleife (benannt nach dem Schlüsselwort `while`). Hier ein Beispiel:

```c
/* while_1a.c */
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    while(1)
    {
        printf("Immer dasselbe...");
        fflush(stdout); /* sofortige Ausgabe des Ausgabepuffers erzwingen */
        sleep(1);
    }
    return 0;
}
```

Solange der Ausdruck in den runden Klammern hinter dem Schlüsselwort , die sogenannte Eintrittsbedingung, ungleich null ist, wird die darauffolgende Anweisung (alles bis zum Semikolon) oder der folgende Block (alles innerhalb geschweifter Klammern) ausgeführt. In diesem Fall läuft die Schleife also endlos, und das Programm kann nur mit Strg-C abgebrochen werden. Anders das folgende Programm:

```c
/* while_1b.c */
#include <stdio.h>

int main(void)
{
    while(0)
    {
        printf("Das hier liest nie jemand...\n");
    }
    return 0;
}
```

1.6.2 Eigene Variable für die Eintrittsbedingung

In der Praxis ist die Eintrittsbedingung häufig mit einer eigenen Variablen gefüllt. Im folgenden Beispiel kann damit der Benutzer angeben, wie lange die Schleife durchlaufen werden soll:

```c
/* while_2.c */
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
#define PI 3.14159

int main(void)
{
    int weiter;
```

```
10
11      setlocale(LC_ALL, getenv("LANG"));
12      weiter=1;
13      while(weiter)
14      {
15          double d;
16          double Aq;
17
18          printf("Leiterdurchmesser in mm:");
19          scanf("%lf", &d);
20          Aq=PI/4.0*d*d;
21          printf("Querschnittsflaeche=%lfmm^2\n", Aq);
22          printf("Weiter (Ja=1/Nein=0)?");
23          scanf("%i", &weiter);
24      }
25      return 0;
26  }
```

1.6.3 Arbeits-Variable für die Eintrittsbedingung

Es ist möglich, den Schleifendurchlauf von einer vorhandenen Variable abhängig zu machen; auch hier kann der Benutzer hiermit vorgeben, wie lange die Schleife durchlaufen werden soll:

```
1   /* while_3.c */
2   #include <stdio.h>
3   #define PI 3.14159
4
5   int main(void)
6   {
7       double d=15.0;
8       while(d>=0)
9       {
10          double Aq;
11          printf("Leiterdurchmesser in mm (Ende mit Wert<0):");
12          scanf("%lf", &d);
13          if(d>=0)
14          {
15              Aq=PI/4.0*d*d;
16              printf("Querschnittsflaeche=%lfmm^2\n", Aq);
17          }
18      }
19      return 0;
20  }
```

Negative Werte für den Drahtdurchmesser d kommen nicht vor, also bedeuten sie das Programmende. Diese Lösung ist problematisch, sobald *doch* einmal ein solcher reservierter Wert eingegeben werden soll — z.B. tauchen negative Innenwiderstände, Frequenzen gleich null, Verstärkungen kleiner eins eben *doch* manchmal auf und bewirken dann ein unerwartetes Programmverhalten.

1.6.4 Eintrittsbedingung als Ergebnis fortlaufender Berechnungen

Im folgenden Beispiel soll gerechnet werden, bis die Zahl null erreicht wird:

```
1   /* while_4a.c */
```

```c
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    int zahl;
    zahl = 5;
    while(zahl >= 0)
    {
        printf("%i\n", zahl);
        zahl = zahl - 1;
        sleep(1);
    }
    return 0;
}
```

`zahl=zahl-1` bedeutet: Der Wert für `zahl` wird aus der Variablen entnommen, dieser Wert wird mit eins in eine Subtraktion einbezogen, das Ergebnis wird wieder in die Variable `zahl` geschrieben. Im Endeffekt wird `zahl` also um eins verringert. Auch bei dieser Konstruktion ist Vorsicht angesagt. Hier ist eine Endlosschleife vorhanden:

```c
/* while_4b.c */
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    unsigned int zahl;
    zahl = 5U;
    while(zahl >= 0U)
    {
        printf("%u\n", zahl);
        zahl = zahl - 1U;
        sleep(1);
    }
    return 0;
}
```

Der Wert für Zahl kann nie negativ sein (Datentyp `unsigned int`), daher ist die Eintrittsbedingung immer erfüllt.

1.6.5 Berechnungen im Schleifenrumpf

Das folgende Programm zeigt die Berechnung einer Summe in einer Schleife.

```c
/* while_5a.c */
#include <stdio.h>

int main(void)
{
    int preis=1; /* irgendein Wert groesser null */
    int summe=0;
    while(preis>=0)
    {
        printf("Stueckpreis in ct (Ende mit Wert<0):");
```

```
11          scanf("%i", &preis);
12          if(preis>=0)
13          {
14              summe = summe + preis;
15              printf("Summe = %i ct\n", summe);
16          }
17      }
18      return 0;
19  }
```

Wie kommt man auf diesen Programmtext? Anhand des folgenden (einfacheren) Beispiels sollen Schritte dazu aufgezeigt werden.

- Geforderte Ausgabe: 1 2 3 4 5 6 7 ...
- Schritt 1: Lösung als einfache Sequenz

```
printf("%d",1);
printf("%d",2);
printf("%d",3);
...
```

- Schritt 2: Variable statt Konstanten verwenden

```
i=1; printf("%d",i);
i=2; printf("%d",i);
i=3; printf("%d",i);
...
```

- Schritt 3: Variable automatisch berechnen

```
i=0;
i=i+1; printf("%d",i);
i=i+1; printf("%d",i);
i=i+1; printf("%d",i);
...
```

- Schritt 4: Immer wiederkehrende Anweisungen in eine Schleife setzen:

```
i=0;
while(1)
{
   i=i+1; printf("%d",i);
}
```

- Schritt 5: Schleifenbedingung wählen, um gegebenenfalls die Anzahl der Durchläufe zu begrenzen:

```
i=0;
while(i<40)
{
   i=i+1; printf("%d",i);
}
```

1.6.6 Gleitkommavariablen und Endlosschleifen

Gleitkommavariablen sollten (wenn irgend möglich) niemals zur Steuerung von Schleifen (oder Verzweigungen) verwendet werden. Die folgende Schleife ist z.B. endlos:

```c
/* while_4c.c */
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    float zahl;
    zahl = 20000000.0F;
    while(zahl >= 0.0F)
    {
        printf("%f\n", zahl);
        zahl = zahl - 1.0F;
        sleep(1);
    }
    return 0;
}
```

Beim Datentyp `float` ist nämlich 20.000.000+1.0 = 20.000.000; bei Gleitkommazahlen bleibt nur die relative Genauigkeit etwa konstant, die absolute Genauigkeit sinkt mit zunehmendem Zahlenwert. Auch die folgende Schleife (20 Durchläufe erwartet) kommt nie an:

```c
/* while_4d.c */
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    float zahl;
    zahl = 16777200.0F;
    while(zahl < 16777220.0F)
    {
        printf("%f\n", zahl);
        zahl = zahl + 1.0F;
        sleep(1);
    }
    return 0;
}
```

Nach 16777215 ist die nächste darstellbare Zahl 16777216, die Zahl danach aber 16777218. Nicht nur das:

```c
/* while_4e.c */
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    float zahl;
    zahl = 16777200.0F;
    while(zahl < 16777217.0F)
    {
```

```
11          printf("%f\n", zahl);
12          zahl = zahl + 1.0F;
13          sleep(1);
14      }
15      return 0;
16  }
```

Die Zahlen 16777216 und 16777217 werden intern gleich dargestellt, sind also nicht unterscheidbar.

```
1   /* while_4f.c */
2   #include <stdio.h>
3   #include <unistd.h>
4   int main(void)
5   {
6       while (16777216.0F == 16777217.0F)
7       {
8           printf("Mathematisch␣sehr␣interessant!\n");
9           sleep(1);
10      }
11      return 0;
12  }
```

1.6.7 `scanf()` und Schleifen

Bei der Benutzung von `scanf()` im Zusammenhang mit Schleifen fällt erstens auf, dass diese Funktion nur so weit liest, wie der Formatstring es vorgibt. Zweitens fällt auf, dass der Rest der Eingabe im Eingabepuffer bleibt. Will man also eine Zahl einlesen lassen und gibt `20XYZ` ein (gefolgt von der ↵-Taste), dann wird nur `20` in die Variable eingelesen, der Rest (`XYZ\n`) bleibt im Puffer und wird vom nächsten `scanf`-Aufruf wieder zu lesen versucht. Bei `scanf`-Aufrufen in einer Schleife kann das zum Problem werden: Gibt man nur einmal statt einer Ziffer einen Buchstaben ein, wird dieser Buchstabe wieder und wieder erfolglos zu lesen versucht.

Für dieses Problem gibt es zwei Lösungsansätze:

a) Die Funktion `scanf()` gibt ein Ergebnis zurück, das man auswerten kann.

b) Man liest einfach den Puffer bis zum Zeilenende leer und wirft alle gelesenen Zeichen weg.

Hier wird mit dem zweiten Ansatz die Eingabe von Zahlen sicherer gemacht:

```
1   scanf("%lf", &x);
2   while(getchar()!='\n'){}
```

In Zeile 1 findet man den ganz normalen Aufruf von `scanf()`. In Zeile 2 wird mit dem Aufruf `getchar()` so lange immer wieder ein Zeichen aus dem Eingabepuffer gelesen, bis man zu einem Zeilenendezeichen kommt. Diese zweite Zeile sollte in Zukunft hinter jeder Benutzung von `scanf()` folgen.