1.2 Programmstrukturen/Berechnungen und Ausgabe

1.2.1 Problem

Der Computer soll den Widerstand einer Parallelschaltung von zwei Widerständen mit $R_1 = 4,7\Omega$ und $R_2 = 6,8\Omega$ berechnen und ausgeben. Zwei Fragen stellt man sich:

- Wie berechnet man eine Zahl in C?
- Wie gibt man eine Zahl in C aus?

1.2.2 Ausgabe von ganzen Zahlen

Bisher ist die Ausgabe von Text bekannt. Dazu diente der Funktionsbaustein `printf`:

```
1 printf("Dies_ist_eine_Speisekarte\n");
```

Aber der `printf`-Baustein kann noch mehr, nämlich Zahlen ausgeben. Die Ausgabe von Zahlen mit `printf` sieht so aus:

```
1 printf("Ein_Glas_Saft_kostet_%i_EUR\n", 4);
```

`%i` ist dabei ein Platzhalter für eine ganze Zahl. 5 ist die Zahl, die ausgegeben wird. Hierbei handelt es sich um eine literale, d.h. wörtliche Zahlenkonstante mit ganzzahligem Typ. Im Kontext sieht es so aus (`zahlausgabe1.c`):

```
1 #include <stdio.h>
2 int main(void)
3 {
4     printf("Dies_ist_eine_Speisekarte.\n");
5     printf("Ein_Glas_Saft_kostet_%i_EUR.\n", 4);
6     return 0;
7 }
```

Den Text mit dem Platzhalter nennt man übrigens *Formatstring*. Die ganze Liste zwischen den beiden Klammern nennt man *Parameterliste*. Die Elemente der Liste werden dabei durch Komma getrennt.

1.2.3 Zahlenkonstanten in C

Zahlen, die direkt im Quelltext stehen, nennt man *literals* (=wörtliche) *Zahlenkonstanten*. Wenn im Quelltext 123 steht, ist damit wie erwartet die Zahl einhundertdreißig gemeint. In C wird nun grundlegend unterschieden zwischen ganzen Zahlen und Gleitkommazahlen.

1.2.3.1 Ganzzahlenkonstanten Es gibt in C mehrere Möglichkeiten, Ganzzahlenkonstanten (=konstante ganze Zahlen) darzustellen:

- Dezimalzahlenkonstanten beginnen mit einer der Ziffern 1 bis 9, danach sind alle Ziffern von 0 bis 9 erlaubt
- Oktalzahlenkonstanten beginnen mit der Ziffer 0, danach sind alle Ziffern von 0 bis 7 erlaubt, nicht jedoch 8 oder 9
- Hexadezimalzahlenkonstanten beginnen mit 0x, danach sind alle Ziffern von 0 bis 9 und alle Buchstaben von a bis f erlaubt (x und a bis f dürfen hier klein- oder großgeschrieben werden)
- Dualzahlenkonstanten (seit C23) beginnen mit 0b, danach sind nur die Ziffern 0 und 1 erlaubt

Zusätzlich kann am Beginn jeder Zahlenkonstante ein Minuszeichen stehen. Dann ist der Wert des Ausdrucks negativ. Tabelle 1 zeigt einige Beispiele.

C-Schreibweise	Wert	Erläuterung
123	$123_{(10)}$	dezimal
0b10101	$10101_{(2)} = 21_{(10)}$	dual (seit C23)
0123	$123_{(8)} = 83_{(10)}$	oktal
0X123	$123_{(16)} = 291_{(10)}$	hexadezimal
0xEF80	$EF80_{(16)} = 61312_{(10)}$	hexadezimal
0xef80	$EF80_{(16)} = 61312_{(10)}$	hexadezimal
-123	$-123_{(10)}$	dezimal, negativ
19	$19_{(10)}$	dezimal
019	—	ungültig: 9 ist keine Oktalziffer
0x1e7	$1E7_{(16)} = 487$	hexadezimal (vergleiche 1e7 unten)

Tabelle 1: Beispiele für Ganzzahlenkonstanten

1.2.3.2 Gleitkommazahlenkonstanten Gleitkommazahlenkonstanten (=konstante Gleitkommazahlen) enthalten dagegen entweder:

- einen Punkt als Dezimaltrenner (Trenner zwischen Vor- und Nachkommaanteil)
- das Zeichen e als Trenner zwischen Mantisse und Exponent
- oder beides

Ist der Vor- oder der Nachkommaanteil null, dann kann man die jeweilige Null weglassen. Bei der Zahl 0,0 muss man allerdings eine der beiden Nullen hinschreiben (sonst bliebe ja nur ein Punkt). Tabelle 2 zeigt wieder einige Beispiele.

C-Schreibweise	Wert	Erläuterung
-2.5	-2,5	Punkt statt Komma
123.	123,0	Null am Ende darf fehlen
-.8	-0,8	Null am Anfang darf fehlen
0.0	0,0	Gleitkomma-Null
0.	0,0	Null am Ende darf fehlen
.0	0,0	Null am Anfang darf fehlen
1e7	$1 \cdot 10^7$	Gleitkommazahl (vergleiche 0x1e7 oben)
2e-1	$2 \cdot 10^{-1} = 0,2$	negativer Exponent
-2e5	$-2 \cdot 10^5 = -200\,000,0$	negative Mantisse
-2e-3	$-2 \cdot 10^{-3} = -0,002$	Mantisse und Exponent negativ
1.45e1	14,5	sowohl Punkt als auch e vorhanden

Tabelle 2: Beispiele für Gleitkommazahlenkonstanten

1.2.4 Ausgabe von Kommazahlen

Bei der Parallelschaltung der beiden Widerstände wird das Ergebnis vermutlich eine Kommazahl sein. Wie kann man nun Kommazahlen ausgeben? Ein Beispiel sieht man hier (zahlausgabe2.c):

```
1 printf("Ein_Glas_Wasser_kostet_%lf_EUR\n", 5.30);
```

Was erkennt man? %lf ist ein Platzhalter für die Ausgabe einer Komma-Zahl. 5.30 meint die Zahl, die ausgegeben wird. Im Kontext sieht die Ausgabe so aus (zahlausgabe2.c):

```
1 #include <stdio.h>
2 int main(void)
```

```

3 {
4     printf("Ein_Glas_Wasser_kostet_%lf_EUR.\n", 5.30);
5     return 0;
6 }

```

1.2.5 Ausgabe mehrerer Zahlen in einem Aufruf

Man kann mit einem Aufruf von `printf()` mehrere Zahlen auf einmal ausgeben:

```

1     printf("Ketchup:%lf_EUR,_Mayonnaise:%lf_EUR\n", 0.5, 0.3);

```

Die Parameterliste hat hier drei Parameter, nämlich den Formatstring und zwei Zahlen. Und im Formatstring sieht man zwei Platzhalter. Wenn der Aufruf von `printf` ausgeführt wird, wird die erste Zahl an die Stelle des ersten Platzhalters gesetzt und die zweite Zahl an die Stelle des zweiten Platzhalters. Auf die gleiche Art kann man auch drei oder mehr Zahlen auf einmal ausgeben:

```

1     printf("Ketchup:%lf_EUR,_Mayonnaise:%lf_EUR,_Senf:%lf_EUR\n",
2         0.5, 0.3, 0.4);

```

1.2.6 Berechnungen

Man kann in C – wie in den meisten Programmiersprachen – auch Zahlenwerte berechnen:

```

1     printf("Pommes_und_Ketchup_kosten_%lf_EUR\n", 2.50 + 0.50);

```

Was ist hier anders? Statt einer Zahlenkonstante steht hier – mitten in der Parameterliste – ein so genannter (mathematischer) *Ausdruck*. In C darf in einer Anweisung immer an der Stelle einer Zahlenkonstante ein Ausdruck stehen.

Ein Ausdruck kann aus *Operatoren* und Zahlenkonstanten bestehen (und aus Variablen und den Ergebnissen von Funktionsaufrufen, dazu später). Das Pluszeichen ist solch ein Operator, genauer gesagt ein Rechenoperator. Mit dem Ausdruck $2.50 + 0.50$ wird das Ergebnis der Rechenoperation $2,5 + 0,5$ ermittelt.

Zuerst wird der Wert (=das Ergebnis) eines Ausdrucks berechnet.¹ Danach wird der berechnete Wert an der Stelle eingesetzt, an der der Ausdruck steht.

Für Kommazahlen gibt es die vier Rechenoperatoren $+$ (plus), $-$ (minus), $*$ (mal) und $/$ (geteilt durch).

Mit ihnen kann man (wenn man will) komplizierte Ausdrücke bilden wie z.B. $5+3$, $3+4*5$, $(7+8)*(2+3)$, $3-2-1$. Die Ausdrücke werden in einer feststehenden Reihenfolge abgearbeitet:

- Was in Klammern steht, wird zuerst bearbeitet.
- Bei verschiedenen Operatoren gibt es eine festgelegte Rangfolge. Es gilt z.B. *Punktrechnung vor Strichrechnung*. Damit ergibt $1+2*3$ den Wert 7.
- Bei mehreren gleichen Operatoren² wird meistens von links nach rechts gearbeitet. So ergibt $7-4-2$ den Wert 1.

¹Falls der Ausdruck nur aus Konstanten besteht, kann das schon bei der Übersetzung des Programmes durch den Compiler geschehen. Im anderen Fall (der Ausdruck enthält auch Variablen oder Ergebnisse von Funktionsaufrufen) passiert das erst zur Laufzeit des Programmes.

²Plus- und Minus-Operator haben den gleichen Rang. Ebenso haben Mal- und Geteilt-durch-Operator den gleichen Rang.

1.2.7 Ganzzahl-Division und Gleitkomma-Division

1.2.7.1 Ganzzahl-Division Für ganze Zahlen gibt es bei der Division noch eine Besonderheit: Der Ausdruck `49/10` ergibt als Ergebnis wieder eine ganze Zahl mit dem Wert 4. Der Nachkommaanteil wird nicht gerundet, sondern fällt einfach weg.

Um den Nachkommaanteil doch noch zu erhalten, gibt es für ganze Zahlen den *Modulo-Operator*, mit dem man den Divisionsrest ermittelt. Der Ausdruck `49%10` (wird gelesen: 49 modulo 10) ergibt als Ergebnis eine ganze Zahl mit dem Wert 9. Das Prozentzeichen hat dabei absolut nichts mit dem Prozentzeichen bei der `printf()`-Funktion zu tun. In der normalen Arithmetik entspricht das dem Wert aus:

$$49 \div 10 = 4 \text{ Rest } 9$$

oder:

$$49 \div 10 = 4 \frac{9}{10}$$

1.2.7.2 Gleitkomma-Division Bei der Division von Gleitkommazahlen ist das Ergebnis wieder eine Gleitkommazahl: Der Ausdruck `49.0/10.0` ergibt wie zu erwarten 4,9 — allerdings mit einer bestimmten relativen Genauigkeit, denn die Zahl 4,9 kann man im Dualzahlensystem nicht exakt darstellen. Der Modulo-Operator ist für Gleitkommazahlen übrigens nicht erlaubt; ein Ausdruck wie `49.0%10.0` wird vom Compiler als ungültig bezeichnet³.

Tabelle 3 fasst das Beschriebene kurz zusammen.

C-Ausdruck	Ergebnis	Erläuterung
<code>49 / 10</code>	4	Ergebnis der Ganzzahl-Division
<code>49 % 10</code>	9	Rest der Ganzzahl-Division
<code>49.0 / 10.0</code>	4.9	Ganzzahl-Division
<code>49.0 % 10.0</code>	—	ungültiger Ausdruck

Tabelle 3: Ganzzahl- und Gleitkomma-Division in C

1.2.8 Symbolische Konstanten

Wenn man eine Zahl oder einen Text mehrmals in einem Programm braucht, möchte man ihn vielleicht nicht jedes Mal detailliert ausschreiben, sondern eine einheitliche Kurzform dafür verwenden. Zu diesem Zweck gibt es in C die Möglichkeit, sogenannte Symbolische Konstanten festzulegen:

```

1 #include <stdio.h>
2 #define WASSERPREIS 5.30
3 #define GRUSS "Guten_Abend!\n"
4 int main(void)
5 {
6     printf(GRUSS);
7     printf("5_Glas_Wasser_kosten_%f_EUR.\n", 5*WASSERPREIS);
8     return 0;
9 }
```

Mit einer Zeile, die mit `#define` beginnt, legt man eine Konstante fest: Das Wort, das auf `#define` folgt, ist der Name der Konstanten; der Rest der Zeile ist der Inhalt, der überall eingesetzt wird, wenn der Name der Konstanten im Quelltext auftaucht. Das kann ein Zahlenwert sein (wie bei `WASSERPREIS`) oder auch ein Text (wie bei `GRUSS`).

³Der Funktionsbaustein `fmod` aus `math.h` bietet aber diese Funktionalität.

1.2.9 Verschönerung der Ausgabe

Bei der Ausgabe von Gleitkommazahlen mit `printf` stört häufig die große Anzahl an Nachkommastellen. Folgende Modifikationen sind möglich:

```
1 printf("Wasserpreis:_%4lf_EUR\n", 2.45);
2 printf("Wasserpreis:_%2lf_EUR\n", 2.45);
3 printf("Wasserpreis:_%5.2lf_EUR\n", 2.45);
```

In Zeile 1 wird die Zahl der auszugebenen Zeichen mit 4 angegeben. In Zeile 2 wird die Zahl der Nachkommastellen mit 2 angegeben. In Zeile 3 legt man fest: Die Gesamtzahl der auszugebenen Zeichen soll 5 sein, davon 2 Nachkommastellen. Gleitkommazahlen kann man (wie bisher) in Festkommadarstellung, aber auch in Gleitkommadarstellung ausgeben:

```
1 printf("Widerstand_R1=%lf_Ohm\n", 100e3);
2 printf("Kapazitaet_C1=%le_F\n", 100e-12);
3 printf("Induktivitaet_L1=%lg_H\n", 100e-6);
```

Der Platzhalter `%lf` (Zeile 1) gibt eine Gleitkommazahl in Festkommadarstellung aus. Der Platzhalter `%le` (Zeile 2) wählt die Gleitkommadarstellung, egal, wie groß die dargestellte Zahl ist. Mit dem Platzhalter `%lg` (Zeile 3) schließlich werden betragsmäßig sehr große und sehr kleine Zahlen in Gleitkommadarstellung, betragsmäßig mittelgroße Zahlen in Festkommadarstellung ausgegeben, was im täglichen Leben auch üblich ist.

Auch bei ganzen Zahlen kann man die Ausgabe anpassen:

```
1 printf("Einwohnerzahl_Pitcairn:_%5d<\n", 51);
2 printf("Einwohnerzahl_Pitcairn:_%05d<\n", 51);
3 printf("Einwohnerzahl_Pitcairn:_%5.3d<\n", 51);
4 printf("Einwohnerzahl_Pitcairn:_%+d<\n", 51);
```

In Zeile 1 werden 5 Zeichen rechtsbündig ausgegeben; die führenden Zeichen sind Leerzeichen. In Zeile 2 werden die 5 Zeichen durch Nullen aufgefüllt. In der letzten Zeile werden wieder fünf Zeichen ausgegeben; jetzt wird nur bis zur 3. Stelle mit Nullen aufgefüllt. Mit einem Pluszeichen hinter dem Prozentzeichen (Zeile 4) kann man die Ausgabe eines Pluszeichens bei positiven Zahlen erzwingen.

Insgesamt setzt sich ein Platzhalter für `printf` aus bis zu sieben Elementen zusammen:

1. Prozentzeichen: `%`
2. Flags (Schalter): `-, +, 0, _, #`
3. Feldbreite: eine Zahl oder `*`. So viele Zeichen werden mindestens ausgegeben. Falls nötig, wird mit Leerzeichen aufgefüllt (oder auf Wunsch mit führenden Nullen). Steht für die Feldbreite anstelle der Zahl ein Stern (`*`), dann wird der Wert für die Feldbreite dem nächsten Element der Liste entnommen.
4. Punkt: `.`
5. Genauigkeit: eine Zahl. Bei Gleitkommazahlen die Anzahl der Nachkommastellen. Bei Ganzzahlen die Anzahl der mindestens zu schreibenden Ziffern (mit führenden Nullen). Kann auch 0 sein. Steht für die Genauigkeit anstelle der Zahl ein Stern (`*`), dann wird der Wert für die Genauigkeit dem nächsten Element der Liste entnommen.
6. Längenmodifizierer: `h, l` oder `L` – siehe später
7. Umwandlungsbezeichner: `bdeEfgGinospuxX%`

Zeichen	Art	Ergebnis
-	2	Linksbündige Ausgabe
+	2	Positive Zahlen erhalten ein Pluszeichen
0	2	Bis zur Feldbreite werden führende Nullen angezeigt
␣	2	Positive Zahlen erhalten ein Leerzeichen am Anfang
#	2	Alternatives Ausgabeformat
*	3	Die Länge wird dem nächsten Element der Liste entnommen
h, l oder L	6	h=short, l=long, siehe später
i oder d	7	Ganzzahlen: dezimale Ausgabe
b	7	Nichtnegative Ganzzahlen: duale Ausgabe (seit C23)
o	7	Nichtnegative Ganzzahlen: oktale Ausgabe
u	7	Nichtnegative Ganzzahlen: dezimale Ausgabe
x oder X	7	Nichtnegative Ganzzahlen: hexadezimale Ausgabe
f	7	Gleitkommazahlen: normale Ausgabe
e oder E	7	Gleitkommazahlen: Exponentialdarstellung
g oder G	7	Gleitkommazahlen: Ausgabe je nach Wert
c	7	Ausgabe eines Zeichens, siehe später
s	7	Ausgabe einer Zeichenkette, siehe später
p	7	Ausgabe einer Adresse, siehe später
%	7	Ausgabe eines Prozentzeichens
n	7	Eingabe der Anzahl bisher ausgegebener Zeichen, siehe später

Tabelle 4: Flags und Umwandlungsbezeichner für printf

Tabelle 4 zeigt, welche Bedeutungen die einzelnen Flags (Art=2) und Umwandlungsbezeichner (Art=7) haben.

Übrigens: Wenn man in printf einen Text ausgeben will, der ein Prozentzeichen enthält, dann klappt es nicht so einfach:

```

1 #include <stdio.h>
2 int main(void)
3 {
4     printf("Greifen_Sie_zu!_100%-ige_Gewinnchance!\n");
5     return 0;
6 }
```

```

Terminal
schueler@debian964:~$ gcc prozent_falsch.c
schueler@debian964:~$ a.out
Greifen Sie zu! 100-1964723304ge Gewinnchance!
```

Hier erkennt printf im Text den Platzhalter %-i, also Ausgabe einer Ganzzahl, linksbündig. Auch ein Maskieren des Prozentzeichens (\% statt %) nützt nichts. Deshalb gibt es einen Trick: Der Platzhalter % ist der Platzhalter für ein einzelnes Prozentzeichen (siehe Tabelle 4), damit muss man für jedes Prozentzeichen im Text %% schreiben:

```

1 #include <stdio.h>
2 int main(void)
3 {
4     printf("Greifen_Sie_zu!_100%%-ige_Gewinnchance!\n");
5     return 0;
6 }
```

Terminal

```
schueler@debian964:~$ gcc prozent_richtig.c  
schueler@debian964:~$ a.out  
Greifen Sie zu! 100%-ige Gewinnchance!
```