1.5 Programmstrukturen/Verzweigung

1.5.1 Situation

Die Stromstärke I_D durch einen Feldeffekttransistor kann man mit der folgenden Formel berechnen:

$$I_D = \begin{cases} \text{bei} & U_{GS} \geq -U_P & : & I_{D0} \cdot \left(1 - \frac{U_{GS}}{U_P}\right)^2 \\ \text{bei} & U_{GS} < -U_P & : & 0 \end{cases}$$

Hier muss man also – je nach Wert für U_{GS} – eine der beiden Zeilen auswählen und damit I_D ermitteln. Wie kann man das in einem C-Programm verarbeiten?

1.5.2 Verzweigung

Im folgenden Beispiel (`src/hp10.c`) wird die Gebührenberechnung eines fiktiven Internet-Providers dargestellt:

```
1  [B#include <stdio.h>
2
3  int main(void)
4  {
5      int gebuehr;          /* in cent */
6      int speicherplatz;    /* in MB   */
7
8      printf("Wie gross ist die Homepage in MB? ");
9      scanf("%d", &speicherplatz);
10
11     if(speicherplatz<=5)
12     {
13         gebuehr = 499;
14     }
15     else
16     {
17         gebuehr = 1899;
18     }
19
20     printf("Gebuehr betraegt: %d,%02d EUR\n", gebuehr/100, gebuehr%100);
21     return 0;
22 }
```

Der Provider verlangt je nach Größe der Homepage unterschiedliche Gebühren:

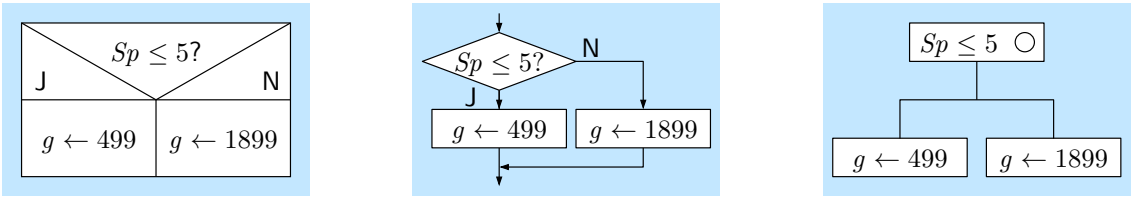
a) Bei einer Datenmenge bis 5 MiB verlangt er 4,99 €

b) Andernfalls verlangt er 18,99 €

Die Programmstruktur, die C hier zur Verfügung stellt, heißt *Verzweigung*. Der Programmablauf wird abhängig gemacht vom Wert der Variable `speicherplatz`. Das Wort `if` ist in C ein *Schlüsselwort*. Es leitet die Verzweigung ein. Der Ausdruck in den runden Klammern direkt hinter dem Schlüsselwort `if` ist der *Bedingungsausdruck*. Hat er einen Wert ungleich null, wird der unmittelbar folgende Block ausgeführt[1]. Man nennt diesen Block *if-Zweig*. Hat der Bedingungsausdruck dagegen den Wert null, dann wird unmittelbar folgende Block direkt hinter dem Schlüsselwort `else` ausgeführt. Er heißt *else-Zweig*. Abbildung 1 zeigt die Darstellung der Verzweigung als Struktogramm, Flussdiagramm und Jackson-Diagramm. Man sieht deutlich, dass immer nur *entweder* der if-Zweig *oder* der else-Zweig ausgeführt wird, niemals beide[2].

[1] Bei manchen Programmiersprachen folgt an dieser Stelle ein Schlüsselwort `then`. In C ist das nicht nötig, weil der Bedingungsausdruck schon durch die runden Klammern begrenzt wird.

[2] weder gleichzeitig noch nacheinander



(a) Struktogramm (b) Flussdiagramm (c) Jackson-Diagramm

Abbildung 1: Verzweigung

1.5.3 Vergleichsoperatoren in C

Der Bedingungsausdruck (also der Ausdruck in runden Klammern hinter dem `if`) ist oft ein Vergleich. Speziell dafür gibt es Vergleichsoperatoren. Sie geben eine Eins zurück, falls der Vergleich zutrifft, ansonsten eine Null. Tabelle 1 gibt stellt sie vor.

mathematisch	C	Ergebnis
$<$	<	falls wahr:1, sonst:0
$>$	>	falls wahr:1, sonst:0
$\leq$	<=	falls wahr:1, sonst:0
$\geq$	>=	falls wahr:1, sonst:0
$=$	==	falls wahr:1, sonst:0
$\neq$	!=	falls wahr:1, sonst:0

Tabelle 1: Vergleichsoperatoren in C

Der Operator zum Prüfen auf Gleichheit ist übrigens „==“. Wenn man stattdessen den Operator „=“ verwendet, nimmt man eine Zuweisung vor:

```
1 if(x=7)   /* falsch, x bekommt hier den Wert 7. */
2 {...}     /* Daher wird hier immer der if-Zweig ausgefuehrt!*/
```

Richtig wäre es so:

```
1 if(x==7)  /* richtig, hier wird x mit 7 verglichen. */
2 {...}
```

1.5.4 Verknüpfungsoperatoren in C

Manchmal möchte man mehrere Vergleiche miteinander verknüpfen, z.B., wenn man abfragen möchte, ob eine Zahl x zwischen 3 und 7 liegt. Dazu gibt es eigene Verknüpfungsoperatoren für UND, ODER und NICHT. Sie sind in Tabelle 2 aufgelistet. Die Abfrage für x sieht dann so aus:

```
1 if(x>=3 && x<=7)
2 {...}
```

mathematisch	C	Ergebnis
$A \wedge B$	A&&B	falls A und B ungleich null:1, sonst:0
$A \vee B$	A||B	falls A oder B ungleich null:1, sonst:0
$\bar{A}$	!A	falls A gleich null:1, sonst:0

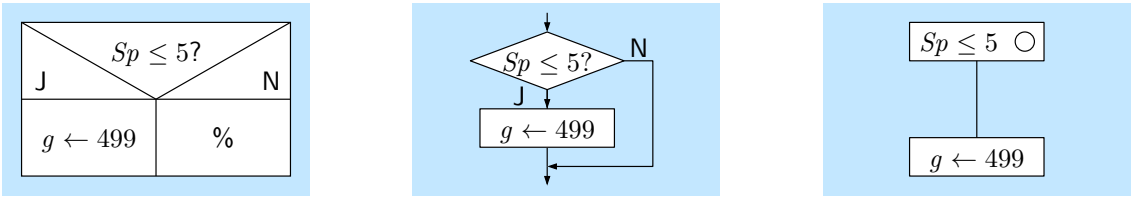
Tabelle 2: Verknüpfungsoperatoren in C

1.5.5 Verzweigung ohne else-Zweig

Wie in den meisten Programmiersprachen darf auch in C der `else`-Zweig einer Verzweigung fehlen. Der Compiler erkennt das am fehlenden Schlüsselwort `else`. Das Beispielprogramm von oben verändert sich dann wie folgt (`src/hp11.c`):

```c
#include <stdio.h>

int main(void)
{
    int gebuehr;          /* in cent */
    int speicherplatz;    /* in MB   */

    printf("Wie gross ist die Homepage in MB? ");
    scanf("%d", &speicherplatz);

    gebuehr = 1899;

    if(speicherplatz<=5)
    {
        gebuehr = 499;
    }
    printf("Gebuehr betraegt: %d,%02d EUR\n", gebuehr/100, gebuehr%100);
    return 0;
}
```

Abbildung 2 zeigt wieder die Darstellung in den verschiedenen Diagrammformen. Im Struktogramm wird der leere else-Zweig durch ein besonderes Zeichen markiert. Das ist wichtig, denn dieses Zeichen sagt aus: „hier ist nichts enthalten“, während ein leerer Struktogramm-Kasten aussagt: „hier ist Inhalt, der nur noch nicht näher bestimmt wurde“.



(a) Struktogramm (b) Flussdiagramm (c) Jackson-Diagramm

Abbildung 2: Verzweigung ohne else-Zweig

1.5.6 Schachteln von Verzweigungen

Jeder Zweig einer if-Anweisung darf wieder beliebig viele weitere Anweisungen, Blöcke und auch if-Anweisungen beinhalten. Ein Beispiel zeigt `src/hp20.c`:

```c
#include <stdio.h>

int main(void)
{
    int gebuehr;          /* in cent */
    int speicherplatz;    /* in MB   */

    printf("Wie gross ist die Homepage in MB? ");
```

```
9     scanf("%d", &speicherplatz);
10
11    if(speicherplatz<=5)
12    {
13        gebuehr = 499;
14    }
15    else
16    {
17        if(speicherplatz<=20)
18        {
19            gebuehr = 1899;
20        }
21        else
22        {
23            gebuehr = 4899;
24        }
25    }
26
27    printf("Gebuehr betraegt: %d,%02d EUR\n", gebuehr/100, gebuehr%100);
28    return 0;
29 }
```

Das entsprechende Struktogramm zeigt Abbildung 3 Hier fällt auf, dass der Quelltext **nach** jeder

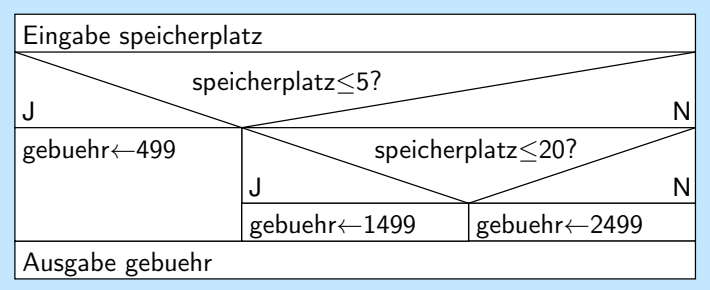


Abbildung 3: Programm mit verschachtelten Verzweigungen

öffnenden geschweiften Klammer um drei Zeichen nach rechts eingerückt wird. **Ab** jeder schließenden geschweiften Klammer geht es wieder um drei Zeichen nach links. Der Compiler berücksichtigt die Einrückung überhaupt nicht; für ihn ist die Formatierung des Quelltextes egal[3] Aber die Einrückung hat *enorme* Vorteile bezüglich der Lesbarkeit eines Quelltextes:

- Man sieht sofort, welche Klammern zusammengehören und von wo bis wo ein Block geht.
- Wenn man eine Klammer vergessen hat, merkt man, dass das Einrücken bis zum Ende des Programms nicht „aufgeht“ und meistens findet man durch das Einrücken den Fehler relativ schnell.

Übrigens kann man bei vielen Editoren das automatische Einrücken einschalten, so dass automatisch jede Zeile so weit eingerückt wird wie die vorherige; einige Editoren verstehen sogar ein wenig C, so dass sie nach jeder geschweiften Klammer automatisch die Einrückung erhöhen oder verringern. Und schließlich gibt es Werkzeuge, die einen Quelltext nachträglich einrücken können[4].

[3] Das ist in den meisten Sprachen so. In der relativ neuen Programmiersprache *Python* dagegen gehört die Einrückung zur Sprache selbst. Eine gute Idee.

[4] Z.B. `bcpp` oder `universalindentgui`

1.5.7 Weglassen der Blockklammern

Im folgenden Quelltext (`src/hp12.c`) sieht man, dass jeder Zweig einer Verzweigung auch statt eines Blockes nur aus einer einzelnen Anweisung bestehen darf. Ein Block ist in C ja schließlich auch eine Anweisung, nur eben eine *Verbundanweisung*.

```c
#include <stdio.h>

int main(void)
{
    int gebuehr;          /* in cent */
    int speicherplatz;    /* in MB   */

    printf("Wie gross ist die Homepage in MB? ");
    scanf("%d", &speicherplatz);

    if(speicherplatz<=5)
        gebuehr = 499;
    else
        gebuehr = 1899;

    printf("Gebuehr betraegt: %d,%02d EUR\n", gebuehr/100, gebuehr%100);
    return 0;
}
```

Mit dieser Einsparmöglichkeit kommen allerdings auch einige Fehlerquellen in die Sprache; so z.B. in `src/hp13defekt.c`:

```c
#include <stdio.h>

int main(void)
{
    int gebuehr;          /* in cent */
    int speicherplatz;    /* in MB   */

    printf("Wie gross ist die Homepage in MB? ");
    scanf("%d", &speicherplatz);

    gebuehr = 1899; /* Standardwert ("Default") */

    if(speicherplatz<=5); /* <==== FEHLER EINGEBAUT! */
        gebuehr = 499;

    printf("Gebuehr betraegt: %d,%02d EUR\n", gebuehr/100, gebuehr%100);
    return 0;
}
```

Das Semikolon hinter dem Bedingungsausdruck ist bereits eine eigene leere Anweisung; somit wird die folgende Zuweisung *immer* ausgeführt. Einen ähnlichen Fehler zeigt `src/hp14defekt.c`:

```c
#include <stdio.h>

int main(void)
{
    int gebuehr;          /* in cent */
    int speicherplatz;    /* in MB   */
```

```
7
8       printf("Wie␣gross␣ist␣die␣Homepage␣in␣MB?␣");
9       scanf("%d", &speicherplatz);
10
11      gebuehr = 1899;
12
13      if(speicherplatz<=5)
14          printf("Sie␣sparen␣bares␣Geld!\n");
15          gebuehr = 499;
16
17      printf("Gebuehr␣betraegt:␣%d,%02d␣EUR\n", gebuehr/100, gebuehr%100);
18      return 0;
19  }
```

Die zweite eingerückte Anweisung hinter der `if`-Zeile scheint zum if-Zweig zu gehören. Das ist aber nur eine optische Täuschung; nur die `printf`-Anweisung gehört dazu. Man muss sich unbedingt merken: Sobald man mehr als eine Anweisung in den if-Zweig holen möchte, braucht man zwingend geschweifte Klammern. Eine dritte Fehlerquelle beim Weglassen der Blockklammern zeigt `src/hp21defekt_warum.c`:

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       int gebuehr;          /* in cent */
6       int speicherplatz;    /* in MB   */
7
8       printf("Wie␣gross␣ist␣die␣Homepage␣in␣MB?␣");
9       scanf("%d", &speicherplatz);
10
11      gebuehr = 1899;
12      if(speicherplatz<=20)    /* if 1 */
13          if(speicherplatz<=5) /* if 2 */
14              gebuehr = 499;
15      else                     /* zu welchem if ?? */
16          gebuehr = 4899;
17
18      printf("Gebuehr␣betraegt:␣%d,%02d␣EUR\n", gebuehr/100, gebuehr%100);
19      return 0;
20  }
```

Es handelt sich um das *dangling-else-Problem*: Bei verschachtelten `if`-Anweisungen kann ein `else` theoretisch zu einer von mehreren `if`-Abfragen gehören; man weiß nur nicht, zu welcher. Deshalb wurde bei C[5] festgelegt: Ein `else` gehört immer zur innersten, also zur zuletzt begonnenen `if`-Anweisung. Und im gezeigten Beispiel ist die Zuordnung, die gemeint und durch die Einrückung auch suggeriert wurde, die falsche.

Aus allen diesen Fehlerquellen folgt die Empfehlung, grundsätzlich immer Blockklammern zu benutzen, es sei denn, der Quelltext würde dadurch wesentlich unübersichtlicher.

[5] und allen anderen Sprachen, die davon betroffen sind