

5.2 Einfache Entscheidungen mit 'if' und die Vergleichsoperatoren

Nun tauchen wir immer tiefer in die Geheimnisse von 'C' ein und beschäftigen uns mit einem sehr wichtigen Kernpunkt: wir treffen **Entscheidungen** und behandeln die sich daraus ergebenden **Konsequenzen** !

Aussagen

Die logischen Aussagen

Die Grundlage für viele #(Entscheidungen)# innerhalb eines Programmablaufes bilden so genannte #(logische Aussagen)#.

MERKE

LOGISCHE AUSSAGEN

Eine (logische) Aussage ist zunächst ein Satz, der entweder wahr oder falsch ist. Der Wahrheitswert muß also eindeutig bestimmbar sein.

Beispiele:

München liegt in Norddeutschland !	ist mit Sicherheit falsch
25 kann ohne Rest durch 5 geteilt werden !	ist wahr
Morgen regnet es vielleicht !	ist keine Aussage im obigen Sinn, denn es kann kein eindeutiger Wahrheitswert zugeordnet werden
Der FC Schalke 04 wird Deutscher Meister !	Sicherlich ein guter Witz, aber keine eindeutig zu bewertende Aussage.

Wichtig ist also, dass (logische) Aussagen grundsätzlich nur zwei Wahrheitszustände haben: sie sind entweder #(wahr)# (#(true)# oder logisch '1') oder sie sind #(falsch)# (#(false)# oder logisch '0'). Ein 'vielleicht' gibt es hierbei nicht !

Aufgrund dieser beiden eindeutigen Zustände können nun innerhalb eines Programms Entscheidungen getroffen werden, die zu ganz unterschiedlichen Programmverläufen führen. Logische Aussagen werden sehr oft auch als #(logische Bedingungen)# bezeichnet. Eine Bedingung kann entweder erfüllt (wahr) oder nicht erfüllt (falsch) sein.

Aus dem täglichen Gebrauch kennen Sie eine Vielzahl solcher Aussagen oder Bedingungen, bei denen es immer nur zwei Wahrheitsalternativen gibt:

BEISPIEL

- Die Tür ist geöffnet: das ist wahr oder falsch.
- Der Motor läuft: das ist wahr oder falsch.

- Der Messwert ist größer 100 V: das ist wahr oder falsch.
- Die Wartezeit ist abgelaufen: das ist wahr oder falsch.

if

.....

*else***Die if...else-Anweisung**

Damit Sie mit dem Wahrheitsgehalt solcher Aussagen den Ablauf eines Mikrocontroller-Programms auch beeinflussen können, gibt es in C die **if...else**-Anweisung, auch **if...else-(Kontrollstruktur)** genannt.

MERKE**DIE if...else-ANWEISUNG**

Die if...else-Anweisung hat ganz allgemein den folgenden Aufbau:

```

if   (Aussage)
{
    (Programmblock bzw. Blockanweisung)
}
else
{
    (Programmblock bzw. Blockanweisung)
}

```

Die if-Anweisung dient also dazu, eine Auswahl **zwischen genau zwei** Möglichkeiten zu treffen:

if...	≡	wenn die nachfolgende
Aussage	≡	wahr ist, dann führe den
Programmblock	≡	nach if aus,
else	≡	sonst (wenn also die Aussage falsch ist) führe
		den
Programmblock	≡	nach else aus.

Man nennt solch einen if-Vergleich auch eine **bedingte (Programm)Anweisung**: Überprüfung einer Bedingung, und je nach deren Wahrheitsgehalt (wahr oder falsch) wird genau einer von zwei bestimmten Programmblöcken ausgeführt.

Ein ganz einfaches Beispiel soll diese if-Konstruktion verdeutlichen:

BEISPIEL

Im nachfolgenden Programmausschnitt aus dem Programm **'if_1.c'** wird eine einzugebende Zahl (0 – 9 ≡ Tastendruck auf der Terminaltastatur) daraufhin verglichen, ob sie größer, gleich oder kleiner als 5 ist (die hier benutzten Vergleichsoperatoren sind nicht allzu schwer zu verstehen und wir werden sie Ihnen im Anschluß sofort erklären):

```
void main (void)
{
    unsigned char i;

    .....
    .....

    // Löschen des Terminal Bildschirms
    printf("\x1b\x48\x1b\x4a");

    printf("Start des Hauptprogramms");

    printf("\n\nBitte eine Taste druecken (0 - 9): ");

    i = getchar();          // Tastendruck einlesen
    i = i - 0x30;           // Tastencode anpassen

    // Erster Vergleich: ist i = 5 ?
    if (i==5)
    {
        printf("\nTaste war 5 !");
    }

    // Zweiter Vergleich: ist i > 5 ?
    if (i>5)
    {
        printf("\nTaste war groesser 5 !");
    }
    // else Zweig: ist i <= 5 ?
    else
    {
        printf("\nTaste war kleiner oder gleich 5 !");
    }

    // Hier: Befehl A;
    // Hier: Befehl B;

}
```

Schauen wir uns dieses Programm nun etwas näher an:

Beim ersten Vergleich wird untersucht, ob i gleich 5 ist. Ist das der Fall, so wird der nach if folgende Programmblock ausgeführt und die entsprechende Meldung erscheint auf dem Bildschirm. Zusätzlich sehen Sie hier, dass der else-Teil bei einer if-Abfrage nicht unbedingt immer vorhanden sein muß.

MERKE**DIE EINFACHE if-ABFRAGE (DER #(EINFACHE if-VERGLEICH)#)**

Beim einfachen if-Vergleich kann der else-Teil entfallen und das bedeutet: ist die Vergleichsaussage wahr, so wird der nach if stehende Programmblock ausgeführt. Ist die Vergleichsaussage dagegen falsch, so wird der gesamte folgende Block übersprungen und mit den danach folgenden Befehlen des Programms weiter fortgefahren.

Beim zweiten Vergleich wird eine vollständige if-Abfrage durchgeführt, also ein if-Vergleich mit else-Teil: ist $i > 5$, so wird der if-Block ausgeführt und bei i nicht größer als 5 der else-Block.

Achtung:
Mathe !

Zu beachten ist hier unbedingt die „**mathematische Logik**“ des Vergleichs: wenn i genau gleich 5 ist, so wird nicht der if-Block des zweiten Vergleichs bearbeitet, sondern der else-Block des zweiten Vergleiches wird ausgeführt, da i ja nicht größer als 5 ist. Bei der hier vorliegenden Programmierung erscheinen also genau zwei Meldungen, wenn Sie auf die Taste '5' drücken, i also genau den Wert 5 hat:

- einmal: „Taste war 5 !“ (vom ersten Vergleich her) und
- einmal: „Taste war kleiner oder gleich 5 !“ (vom else-Teil des zweiten Vergleichs her).

Es ist also auch hier sehr wichtig, dass Sie sich darüber im klaren sind, wie die logischen Aussagen bei einem Vergleich mathematisch exakt interpretiert werden, damit das Programm auch das macht, was **Sie** wollen.

Nach der Durchführung des zweiten Vergleichs in 'if_1.c' wird dann mit den Befehlen A, B, usw. des Programms weiter fortgefahren.

Praxis

- 1) Bei der Programmierung der if-Anweisung sollten Sie sowohl den if- als auch den else-Programmblock im Programmtext einrücken, damit bereits optisch die if...else-Konstruktion klar erkannt wird.
- 2) Wenn der Programmblock aus nur einer einzigen Anweisung besteht, so können die geschweiften Klammern auch entfallen, also:

```
if (i==5)
    printf(".....");
if (i > 5)
    printf(".....");
else
    printf(".....");
// Befehl A;
```

- 3) Man kann die einzelne Anweisung zum if- bzw. zum else-Zweig auch direkt hinter if bzw. else schreiben. So erhält man einen kompakteren Quelltext und die Lesbarkeit bleibt dennoch erhalten:

```
if (i==5) printf(".....");
if (i > 5) printf(".....");
else printf(".....");
```

```
// Befehl A;
```

Vergleiche

Die #(Vergleichsoperatoren)“

Zur Durchführung der Vergleiche bei einer if-Abfrage stehen Ihnen die bekannten #(mathematischen Vergleichsoperatoren)# zur Verfügung.

MERKE

DIE VERGLEICHSOPERATOREN IN C

Zur Durchführung von Vergleichen stehen in 'C' die folgenden Operatoren zur Verfügung:

==	gleich
>	größer
<	kleiner
!=	ungleich
>=	größer gleich
<=	kleiner gleich

Auch hier geht 'C' wieder eigene Wege, insbesondere müssen Sie bei der „Gleichheit“ darauf achten, dass Sie zwei Mal das '='-Zeichen schreiben. Ein einziges '='-Zeichen bedeutet ja eine Variablen-Wert-Zuweisung, und die ist in einem Vergleich völlig fehl am Platz.

Der Compiler antwortet in diesem Fall aber weder mit einer Fehlermeldung noch mit einem Warning, da eine Variablen-Zuweisung ja O.K. ist. Er legt sogar einen Intel-HEX-File an, aber das Programm funktioniert absolut nicht so, wie es soll !

Und diesen Fehler zu finden, bereit einige Mühe, da man die Notwendigkeit von 2 Mal '=' bei einem Vergleich nicht gewohnt ist !

Weiterhin müssen die Vergleichsausdrücke immer in runden Klammern (...) eingeschlossen werden.

Der #(Negationsoperator)# '!'

Durch den Operator '!' wird eine Bedingung negiert, d.h. ihr Wahrheitsgehalt umgekehrt (das entspricht dem logischen #(NOT)#):

```
if ( !( i==3785 ) )
{
    Programmblock
}
```

```
}
```

Der Programmblock wird also ausgeführt, wenn i **nicht gleich** (also ungleich) 3785 ist.

Der Wert von logischen Aussagen

Wie werden Bedingungen (logische Aussagen) nun in 'C' „wertemäßig“ behandelt ?

Es gilt hierbei der folgende Merksatz:

MERKE

DER WERT VON #(BEDINGUNGEN)#

Bedingungen werden 'C'-intern wie ganz normale Zahlenwerte behandelt und zwar gilt:

- eine wahre (positive) Bedingung hat immer den Wert ungleich 0, also z.B. 1
- eine falsche (negative) Bedingung hat immer den Wert 0

Anmerkung: 'C' selber verteilt intern immer nur die beiden Werte 0 oder 1. In Ihrem Programm können Sie aber ganz allgemein mit den Werten 0 und ungleich 0 arbeiten.

Damit können Sie jetzt auch vereinfacht programmieren:

```
if (a)
{
    Dieser Programmblock wird ausgeführt, wenn a ungleich 0
    ist, denn alle Werte ungleich 0 sind in C immer wahr !
}

if (!b)
{
    Dieser Programmblock wird genau dann ausgeführt, wenn
    b = 0 ist, denn (!0) ist jetzt wahr !
}
```

HARRY POTTER LÄBT GRÜßEN ! (ÜBUNG)

Schreiben Sie ein Programm namens 'hermine_1.c' für ein kompliziertes **Code-Schloß** zum Zutritt zur geheimen Geheim-Kammer im Schloß Hogwarts.

Um das Schloß zu öffnen, müssen über die Terminal-Tastatur die Buchstaben des Geheim-Namens in der richtigen Reihenfolge eingegeben werden: „**Hermine**“.

Die gleichen Buchstaben, in einer anderen Reihenfolge eingegeben, sollen das Schloß jedoch **nicht** öffnen !

Auch die Groß – und Kleinschreibung muß richtig beachtet werden !

Das Öffnen des Schlosses wird hier dadurch simuliert, dass Sie eine unsigned char-Variable verwenden, die nur dann einen bestimmten, richtigen Wert enthält, wenn die Eingabebedingungen erfüllt worden sind. Andernfalls soll die Variable einen anderen, falschen Wert

gungen erfüllt worden sind. Andernfalls soll die Variable einen anderen, falschen Wert enthalten.

Weiterhin soll das Öffnen des Schlosses dadurch gekennzeichnet werden, dass auf dem Terminal-Bildschirm die Meldung „Geheimnis gelüftet !“ erscheint, begleitet von einem Beep. Bei einem Fehlversuch erscheint „Zu spät, Lord Valdemort geweckt !“ und ein Beep. Das Code-Schloß startet immer neu durch Druck auf die Reset-Taste des TFH-Core ONE.

Überlegen Sie sich vorab, welche Methoden es geben kann, die geforderten Anforderungen programmtechnisch umzusetzen !