

Kapitel 2

Einfache Beispiel- programme



- 2.1 Ausgabe auf dem Bildschirm
- 2.2 Lokale Variablen, Ausdrücke und Schleifen
- 2.3 Zahlen von der Tastatur einlesen
- 2.4 Formatierung bei der Ausgabe
- 2.5 Zusammenfassung

2 Einfache Beispielprogramme

Mit dem „Hello World“-Programm in Kapitel 1.1 haben Sie bereits erste Erfahrungen im Programmieren gesammelt. Programmieren kann viel Spaß bereiten. Außerdem funktioniert Lernen iterativ. Im Folgenden sollen deshalb weitere kurze und aussagekräftige Programme vorgestellt werden, damit Sie sich spielerisch voranarbeiten, um dann auch Augen und Ohren für die erforderliche Theorie zu haben. Alle Programme des Buches finden sich auch im begleitenden Webauftritt (<http://www.stz-softwaretechnik.de/cbuch/>), sodass Sie die Programme nicht abzutippen brauchen. Es ist in C möglich, bereits mit wenigen einfachen Mitteln sinnvolle Programme zu schreiben. Als Einstieg sollen in den Kapiteln 2.1 bis 2.4 einfache Programmbeispiele vorgestellt werden, um mit **symbolischen Konstanten**, **lokalen Variablen** von Funktionen, **Ausdrücken**, **Schleifen** und der **Ein- und Ausgabe** in C vertraut zu werden.

Bei der Vorstellung der Beispielprogramme werden Sprachmittel wie Präprozessor-Anweisungen (siehe auch Kapitel 21), Funktionen (siehe auch Kapitel 11) und Variablen (siehe auch Kapitel 7) eingeführt, die erst an späterer Stelle ausführlich behandelt werden. Kapitel 2.1 demonstriert die Ausgabe auf dem Bildschirm. Kapitel 2.2 führt symbolische Konstanten, lokale Variablen in Funktionen, Ausdrücke und Schleifen ein. Kapitel 2.3 zeigt, wie ganze Zahlen von der Tastatur eingelesen werden und wieder auf dem Bildschirm ausgegeben werden. Kapitel 2.4 gibt ein Beispiel, wie man Tabellen bei der Ausgabe formatiert.

2.1 Ausgabe auf dem Bildschirm

Es soll ein C-Programm geschrieben werden, das die ersten zehn ganzzahligen Quadratzahlen auf der Konsole ausgibt. Hier das Programm und seine Ausgabe:

```
/* Datei: square.c */ /* das ist ein Kommentarblock */
#include <stdio.h> /* das ist eine Präprozessor-Anweisung */

int main (void) /* das ist das Hauptprogramm */
{
    int lv;
    /* das ist eine Leerzeile */
    printf ("Die ersten zehn ganzzahligen Quadratzahlen sind:\n");
    for (lv = 1; lv <= 10; lv = lv + 1) /* diese for-Schleife */
        printf ("%d ", lv * lv); /* läuft von 1 bis 10 */
    return 0;
}
```



Dieses Programm gibt aus:

Die ersten zehn ganzzahligen Quadratzahlen sind:
1 4 9 16 25 36 49 64 81 100

Die Bestandteile dieses Programms werden im Folgenden Schritt für Schritt vorgestellt und erläutert.

2.1.1 Kommentarzeile

Die erste Zeile des Beispielprogramms enthält einen sogenannten Kommentar, genauer gesagt einen Kommentarblock¹⁵. Als Kommentar gilt in C alles, was zwischen den Zeichenfolgen `/*` und `*/` steht. Ein Kommentar dient zur Dokumentation eines Programms und hat keinen Einfluss auf den Ablauf des Programms. Alle Beispiele in diesem Buch enthalten am Anfang einen Kommentarblock mit dem Dateinamen des entsprechenden C-Programms auf dem begleitenden Webaufttritt.

2.1.2 Include-Anweisung

Die Präprozessor-Anweisung `#include <stdio.h>` ist erforderlich, weil im Programm die Bibliotheksfunktion `printf()` benutzt wird. Zum Einbinden von Bibliotheken folgt in Kapitel 2.1.10 noch ein ausführlicher Abschnitt.

2.1.3 Leerzeilen

Leerzeilen haben keine spezielle Bedeutung in C. Sie können aber ähnlich wie Kommentare dazu benutzt werden, ein Programm optisch zu gliedern.

2.1.4 Die Funktion `main()`

Ein **Hauptprogramm** muss immer vorhanden sein. Mit dem Hauptprogramm **beginnt ein Programm seine Ausführung**. In C heißt das Hauptprogramm stets `main()`.



Die Funktion `main()` wird im Detail in Kapitel 17.1 behandelt. In diesem Kapitel wird der folgende, einfache Kopf der Funktion `main()` benutzt:

```
int main (void)
```

Der Rückgabotyp `int` der Funktion `main()` kennzeichnet, dass die Funktion `main()` einen ganzzahligen Wert an den Aufrufer zurückgeben soll. Die Funktion `main()` soll an den Aufrufer den Status in Form einer ganzen Zahl zurückliefern, ob bei ihrer Abarbeitung alles glatt gegangen ist oder ob Fehler aufgetreten sind. Üblicherweise wird der Wert 0 zurückgegeben, wenn das Programm fehlerfrei abgearbeitet werden konnte.



Der Rückgabewert der Funktion `main()` wird an das aufrufende Programm (beispielsweise eine Kommando-prozedur) zurückgegeben. Dieses übergeordnete Programm kann anhand dieses Codes auf die Beendigung von `main()` reagieren. Auch in der Parallelverarbeitung werden Prozesse anhand der jeweils an den Steuerprozess zurückgegebenen Werte gesteuert. In Kapitel 17.2.2 wird erläutert, wie der Rückgabewert eines Programmes in einer Kommando-prozedur abgeholt und interpretiert werden kann.

¹⁵ C99 und C11 kennt auch Zeilenkommentare, die durch `//` eingeleitet werden.

Innerhalb der runden Klammern kann eine Funktion eine Liste von Parametern¹⁶, die **Parameterliste**, definieren. Diese Parameter werden beim Aufruf der Funktion mit den übergebenen Argumenten¹⁷ gefüllt.

Das Schlüsselwort¹⁸ `void` in der Parameterliste bedeutet, dass eine Funktion **keine Argumente** erwartet.



Es ist auch möglich, beim Aufruf eines Programms Argumente an das Hauptprogramm `main()` zu übergeben. Die Übergabe von Argumenten an `main()` beim Programmaufruf wird in Kapitel 17.1 behandelt.

In C sind die Klammern der Parameterliste bei allen Funktionen erforderlich, selbst wenn es keine Parameter gibt. Dann verwendet man `void` in runden Klammern für eine leere Parameterliste.



Dass Klammern erforderlich sind, liegt daran, dass es in C kein Schlüsselwort für eine Funktion wie etwa `function` gibt. In C charakterisieren die runden Klammern nach dem Funktionsnamen eine Funktion¹⁹. Es ist daher üblich, dass man auch im Text die runden Klammern zu einem Funktionsnamen hinzufügt, um dem Leser anzuzeigen, dass es sich nicht um einen Variablennamen, sondern eben um einen Funktionsnamen handelt.

2.1.5 Geschweifte Klammern

Zwischen den geschweiften Klammern stehen die Anweisungen der Funktion `main()`. Im Allgemeinen werden zur Abarbeitung der Anweisungen Variablen benötigt, um Zwischenergebnisse abzulegen oder um beispielsweise – wie im Falle der Variablen `lv` der in Kapitel 2.1 gezeigten Datei `square.c` – die Anzahl der Schritte mitzuzählen und das Ende der Iteration zu prüfen. Die geschweiften Klammern umschließen in C einen sogenannten **Block** (siehe Kapitel 10.1), der die Definition lokaler Variablen und Anweisungen beinhalten kann. Schreibt man Programme aus mehreren Funktionen, so ruft die Funktion `main()` auch andere Funktionen auf.

Innerhalb der geschweiften Klammern `{ }` stehen die sogenannten Definitionen lokaler Variablen und die Anweisungen einer Funktion. Durch die Definition einer lokalen Variablen wird die lokale Variable²⁰ erzeugt. In C90 müssen zuerst alle Definitionen der Variablen aufgeführt werden, dann erst dürfen die Anweisungen kommen. Seit dem C99-Standard existiert diese Einschränkung nicht mehr.



¹⁶ Diese Parameter werden auch formale Parameter genannt.

¹⁷ Die Argumente werden auch aktuelle Parameter genannt.

¹⁸ Ein Schlüsselwort ist ein in einer Programmiersprache reserviertes Wort mit einer für diese Programmiersprache speziell vorgegebenen Bedeutung. Der Compiler kennt alle Schlüsselwörter. Schlüsselwörter dürfen nicht als eigene Namen z. B. von Typen, Variablen oder Funktionen verwendet werden.

¹⁹ Natürlich gibt es runde Klammern auch bei Schlüsselwörtern wie `for`, `while` oder `if`.

²⁰ Lokale Variablen sind funktionslokal.

2.1.6 Strichpunkt

Ein Semikolon ist in C ein Trenner. Es wird insbesondere dazu verwendet, das Ende einer Anweisung oder einer Definition anzuzeigen.

Eine Anweisung oder eine Definition kann sich über mehrere Zeilen erstrecken.



2.1.7 Definitionen und Anweisungen

In der Datei `square.c` von Kapitel 2.1 kommt zuerst die Definition der ganzzahligen Variablen `lv`:

```
int lv; // hier wird die Variable lv angelegt
```

Dann folgen die Anweisungen:

```
printf ("Die ersten zehn ganzzahligen Quadratzahlen sind:\n");
for (lv = 1; lv <= 10; lv = lv + 1)
    printf ("%d ", lv * lv);
return 0;
```

Der Aufruf von `printf()` und die `return`-Anweisung in der letzten Zeile wurden schon im Zusammenhang mit dem „Hello World“-Programm in Kapitel 1.1 besprochen. Im Folgenden werden die zwei Aufrufe von `printf()` in diesem Beispiel genauer betrachtet. Danach folgt eine Erklärung der hier verwendeten `for`-Schleife.

2.1.8 Die Funktion `printf()`

Mit Hilfe der Funktion `printf()`, deren Schnittstelle in der Header-Datei `<stdio.h>` zu finden ist, wird eine Bildschirmausgabe erzeugt. Beim ersten Aufruf von `printf()` im Beispiel von Kapitel 2.1.7 wird der Funktion `printf()` als Argument die konstante Zeichenkette

```
"Die ersten zehn ganzzahligen Quadratzahlen sind:\n"
```

übergeben. Am Schluss der Zeichenkette steht das **Steuerzeichen** `\n`. Mit diesem Steuerzeichen wird die **Schreibmarke** (der **Cursor**) des Bildschirms an den Beginn der nächsten Bildschirmzeile positioniert. Steuerzeichen werden ausführlich in Kapitel 6 besprochen.

Beim zweiten Aufruf der Funktion `printf()` werden der Funktion `printf()` zwei Argumente übergeben, eine konstante Zeichenkette und der Wert des Ausdrucks `lv * lv`. In der Zeichenkette ist ein sogenanntes Formatelement (`%d`) zu finden, das angibt, in welcher Form das zweite Argument `lv * lv` auszugeben ist. Im konkreten Beispiel bedeutet `"%d "`, dass als erstes ein ganzzahliger Wert in Dezimaldarstellung (`%d`) auszugeben ist und dass danach ein Leerzeichen folgen soll. Weitere Beispiele und Erklärungen zu der Funktion `printf()` und den Formatelementen sind in den folgenden Beispielprogrammen zu finden. Eine ausführliche Beschreibung der Funktion `printf()` findet sich in Kapitel 16.6.2.1.

Mit Hilfe des Aufrufes der Funktion `printf()` wird somit der Wert ausgegeben, der durch den Ausdruck `lv * lv` während eines Schleifendurchlaufs berechnet wird.

2.1.9 for-Schleife

Umgangssprachlich ausgedrückt lautet die C-Anweisung

```
for (lv = 1; lv <= 10; lv = lv + 1)
    printf ("%d ", lv * lv);
```

in etwa so:

„Setze vor dem ersten Durchlauf der Schleife die Zählvariable (Schleifenvariable) `lv` auf den Wert 1 (`lv = 1`). Erhöhe nach jedem Durchlauf den Wert von `lv` um 1 (`lv = lv + 1`). Prüfe vor jedem Durchlauf, ob `lv` einen Wert kleiner gleich 10 hat (`lv <= 10`). Ist die Prüfung wahr, führe den Durchlauf aus, ansonsten breche die Schleife ab und fahre mit der nächsten Anweisung hinter der Schleife fort.“

Durch die `for`-Schleife in dem aufgeführten Beispiel wird bei jedem Schleifendurchlauf genau eine einzige Anweisung ausgeführt, nämlich der Aufruf der Funktion `printf()`. Sollen mehrere Anweisungen während eines Schleifendurchlaufs ausgeführt werden, ist unbedingt darauf zu achten, dass geschweifte Klammern benutzt werden, um einen Block²¹ von Anweisungen zu bilden.

2.1.10 Inkludieren von Bibliotheksfunktionen

Die Sprache C selbst besitzt keine eingebauten Funktionen für die Ein- und Ausgabe am Bildschirm. In C werden hierfür Funktionen mit standardisierten Schnittstellen in sogenannten Standardbibliotheken bereitgestellt. Die Schnittstellen der Funktionen stehen in sogenannten Header-Dateien. Durch Einbinden der entsprechenden Header-Datei in das eigene Programm ist es dem Programmierer möglich, die Ein- und Ausgabefunktionen der Bibliotheken zu nutzen.

Mit Hilfe der `#include`-Anweisung an den Präprozessor

```
#include
```

ist es möglich, eine externe Datei (auch **Header-Datei**, **h-Datei** oder **header file** genannt) für die Dauer des Übersetzungslaufs in den Quelltext zu kopieren. Eine Header-Datei enthält die Schnittstellen der Funktionen, d. h. Funktionsname, Übergabeparameter und Rückgabtyp, die in einer Bibliothek enthalten sind. Damit ist es möglich, Standardfunktionen in einem Programm zu verwenden.



Header-Dateien dienen der Modularisierung. Ihr Inhalt steht allen Quelldateien zur Verfügung und kann bei Bedarf mit `#include` aufgenommen werden.



²¹ Siehe Kapitel 8.1.

Ein typisches Beispiel für eine Anweisung an den Präprozessor ist die Präprozessor-Anweisung `#include <stdio.h>`. Damit wird die Header-Datei `stdio.h` der Standardbibliothek für die Ein- und Ausgabe eingefügt, in welcher sich unter anderem die Schnittstelle der Funktion `printf()` befindet. Erst dadurch wird ein Aufruf der Funktion `printf()` möglich.

Eine Quelldatei mit ihren Include-Dateien stellt eine **Übersetzungseinheit** dar, die getrennt kompiliert werden kann.



Da eine Bibliotheksfunktion vor ihrem Aufruf dem Compiler bekannt gemacht werden muss, sollte man die `#include`-Anweisungen stets an den Anfang einer Quelldatei stellen.



Kapitel 21.2 behandelt Header-Dateien.

2.2 Lokale Variablen, Ausdrücke und Schleifen

An dieser Stelle wird das berühmte Temperaturwandlungsprogramm von Kernighan und Ritchie vorgestellt. Es soll eine Temperatortabelle zur Umrechnung von der Einheit Fahrenheit in die Einheit Celsius erzeugen. Dieses Programm vermittelt tiefere Erfahrungen mit einer **Schleife** und mit der Berechnung von **Ausdrücken**. Es werden drei verschiedene Varianten des Programms vorgestellt.

2.2.1 Variante mit symbolischen Konstanten und int-Variablen

In der ersten Variante dieses Programms werden **symbolische Konstanten**²² für die untere Grenze, die obere Grenze und die Schrittweite in Fahrenheit verwendet. Für die Temperatur in Celsius und Fahrenheit werden `int`-Variablen verwendet, um für verschiedene Werte in Fahrenheit jeweils den entsprechenden Celsius-Wert zu berechnen. Als Schleife wird die `while`-Schleife eingeführt. Hier das Programm:

```
/* Datei: Fahrenheit.c */
#include <stdio.h>

// Hier werden Zeilenkommentare verwendet
// Konstanten
#define UPPER    300      // obere Grenze UPPER ist eine
                          // symbolische Konstante
                          // 300 ist eine literale Konstante
#define LOWER    0        // untere Grenze
#define STEP     20       // Schrittweite

int main (void)
{
    // Variablen
    int fahr;              // Definition der lokalen Variablen fahr
```

²² Symbolische Konstanten sind Konstanten, die einen Namen tragen. An die Stelle eines Namens setzt der Compiler dann die der symbolischen Konstanten zugeordnete literale Konstante ein, also z. B. eine „nackte Zahl“ wie die Zahl 10.

```

// für die Temperatur in der Einheit
// Fahrenheit
int celsius; // Definition der lokalen Variablen celsius
// für die Temperatur in der Einheit
// Celsius

// Anweisungen
fahr = LOWER; // als Anfangswert wird fahr der Wert
// 0 zugewiesen
while (fahr <= UPPER ) // wiederhole, solange fahr kleiner
{ // oder gleich UPPER ist
    celsius = 5 * (fahr - 32) / 9;
    // nach dieser Formel berechnet sich der
    // Celsius-Wert aus einem Fahrenheit-Wert
    printf ("%d\t%d\n", fahr, celsius);
    // Es werden die Werte der Variablen fahr
    // und celsius als Dezimalzahlen
    // ausgegeben. Die Werte werden getrennt
    // durch ein Tabulator-Zeichen und am
    // Ende wird ein Zeilenumbruch ausgegeben
    fahr = fahr + STEP; // Der nächste Wert von fahr wird
    // berechnet
}
return 0;
}

```



Dieses Programm gibt aus:

0	-17
20	-6
40	4
60	15
80	26
100	37
120	48
140	60
160	71
180	82
200	93
220	104
240	115
260	126
280	137
300	148

Durch die Verwendung des Tabulator-Zeichens als Trennzeichen zwischen den Werten beginnen die Celsius-Werte alle in derselben Spalte, unabhängig davon, ob der entsprechende Fahrenheit-Wert aus einer oder mehreren Ziffern besteht.

Die Größen `LOWER`, `UPPER` und `STEP` sind **symbolische Konstanten**.

Namen symbolischer Konstanten werden **üblicherweise in Großbuchstaben** geschrieben.



Bitte beachten Sie:

- Die Division $5/9$ ergibt null (**ganzzahlige Division ohne Rest**). Daher wird die 5 zunächst mit $(fahr - 32)$ multipliziert, damit vor der Division eine große Zahl entsteht. Im zweiten Schritt wird dann durch 9 geteilt.
- Der **Zuweisungsoperator** ist das Zeichen `=`. Für den **Vergleichsoperator** „ist gleich“ muss in C die Notation `==` verwendet werden.
- Eine `while`-Schleife wird abgearbeitet, solange die Bedingung `fahr <= UPPER` wahr ist.
- Mit `printf()` kann nicht nur – wie im Falle von `"Hello, world"` – Text ausgegeben werden, sondern es können auch die Werte von Variablen ausgegeben werden, ja sogar von Ausdrücken wie $3 * 4 + 7$.

2.2.2 Variante ohne symbolische Konstanten

Im Folgenden werden einige andere Varianten dieses Programms vorgestellt. Dabei soll in der nächsten Variante **ohne symbolische Konstanten** und nur mit **Integer-Größen**, d. h. mit **ganzzahligen literalen Konstanten** – also nackten Zahlen –, **ganzzahligen Variablen** und **ganzzahligen Ausdrücken**, gearbeitet werden. Als Schleife wird die schon bekannte `for`-Schleife verwendet. Hier die zweite Variante des Programms:

```
/* Datei: Fahrenheit2.c */
#include <stdio.h>

int main (void)
{
    // Variablen
    int fahr;

    // Anweisungen
    for (fahr = 0; fahr <= 300; fahr = fahr + 20)
    {
        printf ("%d\t%d\n", fahr, 5 * (fahr - 32) / 9);
    }

    return 0;
}
```

Beachten Sie hierbei die folgenden Punkte:

- In der `for`-Schleife stellt `fahr = 0` den Beginn der Schleife dar, `fahr <= 300` die Bedingung, wie lange die Schleife durchgeführt wird, und `fahr = fahr + 20` den nächsten Wert, für den die Schleife durchgeführt wird.
- Die Verwendung des Ausdrucks $5 * (fahr - 32) / 9$ in der `printf()`-Funktion anstelle der Variablen `celsius` wie im vorherigen Beispiel ist beispielhaft für die allgemeine Regel:

In jedem Zusammenhang, in dem der **Wert einer Variablen** eines bestimmten Typs stehen kann, kann **auch ein komplizierter Ausdruck** von diesem Typ stehen.



2.2.3 Variante mit einer double-Variablen

In der letzten Variante soll mit `celsius` als **double-Variable** und mit einer **while-Schleife** gearbeitet werden:

```
/* Datei: Fahrenheit3.c */

#include <stdio.h>

// Konstanten
#define UPPER 300      // obere Grenze
#define LOWER 0        // untere Grenze
#define STEP 20        // Schrittweite

int main (void)
{
    /* Variablen */
    int fahr;
    double celsius;
    // Anweisungen
    fahr = LOWER;
    while (fahr <= UPPER)
    {
        celsius = (5.0 / 9) * (fahr - 32);
        printf ("%d\t%f\n", fahr, celsius);
        fahr = fahr + STEP;
    }
    return 0;
}
```



Dieses Programm gibt aus:

0	-17.777779
20	-6.666667
40	4.444445
60	15.555556
80	26.666668
100	37.777779
120	48.888893
140	60.000004
160	71.111115
180	82.222229
200	93.333336
220	104.444450
240	115.555557
260	126.666672
280	137.777786
300	148.888901

Beachten Sie bitte:

- Die Konstante 5.0 ist vom Typ `double`. Damit ist

`(5.0 / 9) * (fahr - 32)`

vom Typ `double` – der Compiler muss die Zahl 9 und den Ausdruck `(fahr - 32)` ohne eine Anweisung des Programmierers **implizit** in den Gleitpunkt-Typ

C als erste Programmiersprache

Mit den Konzepten von C11

Goll, J.; Dausmann, M.

2014, XX, 727 S. 151 Abb., Softcover

ISBN: 978-3-8348-1858-4