2 Einfache Beispielprogramme

Mit dem „Hello World“-Programm in Kapitel → 1.1 wurden 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, sodass Sie die Programme nicht abzutippen brauchen.

```
https://link.springer.com/
```

Es ist in C möglich, bereits mit wenigen einfachen Mitteln sinnvolle Programme zu schreiben. Als Einstieg sollen in den folgenden Kapiteln einfache Programmbeispiele vorgestellt werden, um mit Konstanten, Variablen, Funktionen, Ausdrücken, Schleifen und der Ein- und Ausgabe in C vertraut zu werden.

2.1 Quadratzahlen

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

square.c

```
#include <stdio.h>

int main(void) {
  int max = 10;

  printf("Die ersten %d ganzzahligen Quadratzahlen sind:\n", max);

  int zahl = 1;
  while (zahl <= max) {
    printf("%d ", zahl * zahl);
    zahl = zahl + 1;
  }

  return 0;
}
```

Ergänzende Information Die elektronische Version dieses Kapitels enthält Zusatzmaterial, auf das über folgenden Link zugegriffen werden kann https://doi.org/10.1007/978-3-658-45209-4_2.

J. Goll und T. Stamm, *C als erste Programmiersprache*,
https://doi.org/10.1007/978-3-658-45209-4_2

Dieses Programm gibt aus:

```
Die ersten 10 ganzzahligen Quadratzahlen sind:
1 4 9 16 25 36 49 64 81 100
```

Kurz zusammengefasst passiert in diesem Programm das Folgende:

Zuerst wird genauso wie im „Hello World“ Beispiel aus Kapitel → 1.1 die Bibliothek `<stdio.h>` eingebunden, womit die Ausgabe auf der Konsole ermöglicht wird. Dann beginnt das Hauptprogramm mit der Funktion `main()`, welche als erstes den Wert `max` definiert. Dann wird ein Text auf der Konsole ausgegeben. Danach startet eine Schleife, innerhalb welcher eine Zahl sukzessive von `1` bis `max` hochgezählt und der quadratische Wert dieser Zahl auf der Konsole ausgegeben wird. Nach Ablauf der Schleife wird das Programm beendet.

Anhand dieses einfachen Beispiels werden hier nun einige grundlegende Elemente der Sprache C erläutert:

2.1.1 Leerzeichen und Leerzeilen

Leerzeichen wie `Space` und `Tab`, sowie Leerzeilen mit `Enter` werden im Englischen als „**whitespace characters**“ bezeichnet und haben keine spezielle Bedeutung in C. Sie können aber dazu benutzt werden, ein Programm optisch zu gliedern.

Die Verwendung von Leerzeichen zur Gliederung des Codes am linken Rand wird „**Einrückung**“, oder auf Englisch „**indentation**“, genannt. Üblich sind 2 oder 4 Leerzeichen pro Einrückungsebene.

Leere Zeilen werden üblicherweise einfach für mehr Abstand zwischen zusammengehörigen Codeteilen verwendet.

2.1.2 Variablen, Definitionen und Anweisungen

Damit ein Programm Berechnungen ausführen kann, benötigt es zum einen einen Platz, um Werte abzurufen und Resultate zu speichern, zum anderen benötigt es Anweisungen, was genau berechnet werden soll.

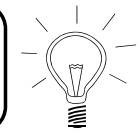
Werte finden in C in sogenannten „**Variablen**“ Platz, welche mittels **Definitionen** erstellt werden. Im obigen Beispiel wird die Variable `max` als Ganzzahl definiert. In der Programmierung wird eine **Ganzzahl** als ein sogenannter **„Integer“** (`int`) bezeichnet. Der Integer `max` wird sodann auch gleich mit dem Wert `10` initialisiert:

```
int max = 10;
```

Danach folgen **Anweisungen** wie die Ausgabe auf dem Bildschirm und die Durchführung der Schleife.

Da Leerzeichen in C keine spezielle Bedeutung haben, benötigt die Sprache ein Trennzeichen, um das Ende einer Anweisung oder einer Definition anzuzeigen. Diese Aufgabe übernimmt in der Sprache C das **Semikolon** (zu Deutsch Strichpunkt).

> Eine Anweisung oder eine Definition wird mittels eines Semikolons ; abgeschlossen.



Beachten Sie, dass das Zeichen = in der Programmiersprache C ein Zuweisungs-Operator ist, sprich der Wert auf der rechten Seite des Gleichheitszeichens wird der Variable auf der linken Seite zugewiesen. Für den Vergleichs-Operator „ist gleich“ muss in C die Notation == verwendet werden.

2.1.3 `while`-Schleife

Im Beispiel wird die folgende Schleife ausgeführt:

```
int zahl = 1;
while (zahl <= max) {
  printf("%d ", zahl * zahl);
  zahl += 1;
}
```

Umgangssprachlich könnte dies in etwa so ausgedrückt werden:

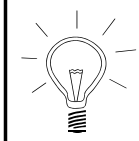
„Setze vor dem ersten Durchlauf der Schleife die Variable `zahl` auf den Wert `1`. Berechne innerhalb der Schleife das Quadrat von `zahl` und gib es auf der Konsole aus. Danach wird `zahl` um `1` erhöht. Führe dies solange durch, wie die Zahl kleiner oder gleich ist wie `max`."

Da bei dieser Schleife mehrere Anweisungen während eines Durchlaufs ausgeführt werden müssen, werden geschweifte Klammern benutzt.

2.1.4 Blöcke und geschweifte Klammern `{}`

Blöcke werden im Detail im Kapitel → 10.1 behandelt. Hier genügt es zu sagen, dass mittels geschweifter Klammern mehrere Anweisungen und Definitionen zusammengefasst werden können.

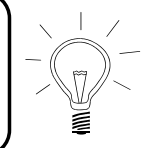
> Geschweifte Klammern umschließen in C einen sogenannten **Block**, der die Definition beliebig vieler lokaler Variablen und Anweisungen beinhalten kann. Lokale Variablen sind nur innerhalb des umschließenden Blocks gültig.



So werden im obigen Beispiel die geschweiften Klammern nicht nur für die `while`-Schleife genutzt, sondern auch für die `main()` Funktion. Die Klammern der `main()` Funktion umfassen sämtliche Anweisungen und Definitionen, welche bei Aufruf ausgeführt werden sollen.

2.1.5 Die Funktion `main()`

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



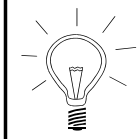
Die Funktion **`main()`** wird im Detail in Kapitel → 17.1 behandelt. In diesem Kapitel wird der folgende, einfache Kopf der Funktion `main()` benutzt:

```
int main()
```

Der Rückgabetyp `int` der Funktion `main()` kennzeichnet, dass die Funktion `main()` einen Integer-Wert an den Aufrufer zurückgeben soll. Tatsächlich findet dies in der letzten Zeile der Funktion mittels `return 0` statt. Die Funktion gibt mit der Zahl `0` an, dass bei ihrer Abarbeitung alles glatt gegangen ist. Weitere Informationen über diesen sogenannten „Status"-Wert können in Kapitel → 17.2 nachgelesen werden.

Es ist auch möglich, beim Aufruf eines Programms Argumente an das Hauptprogramm `main()` zu übergeben. Innerhalb der runden Klammern kann eine Funktion eine Liste von **Parametern**, die sogenannte „Parameterliste", definieren. Diese Parameter werden beim Aufruf der Funktion mit den übergebenen **Argumenten** gefüllt.

Um in C einen Name als Namen einer Funktion zu markieren, sind runde Klammern erforderlich. Diese Klammern müssen geschrieben werden, selbst wenn es keine Parameter gibt. Falls keine Parameter vorhanden sind, kann einfach die Parameterliste leergelassen werden.



Um dem Compiler explizit mitzuteilen, dass keine Parameter erwartet werden, kann auch das Schlüsselwort `void` in die Klammern geschrieben werden:

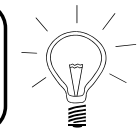
```
int main(void)
```

Das Fehlen dieser Angabe wurde bisher von vielen Compilern mit einer Warnung vermerkt. Ab dem Standard C23 ist dies jedoch unproblematisch.

2.1.6 Die Funktion `printf()`

Wie bereits in Kapitel → 1.1 erwähnt, bedeutet `printf` „print formatted“, also formatierte Ausgabe. Eine Ausgabe wird formatiert, indem in dem übergebenen String eincodiert wird, welche Werte wo und auf welche Weise ausgegeben werden sollen.

> Der an `printf()` übergebene String wird **Format-String** genannt.

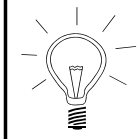


Hier in diesem Beispiel treten zwei **Formatelemente** auf: Die Zeichenfolgen `\n` und `%d`.

Mittels der Zeichenfolge `\n` wird eine neue Zeile (auf Englisch **newline**) begonnen, sprich die nachfolgenden Ausgaben werden auf der nächsten Bildschirmzeile links in der Konsole stehen. Eine Zeichenfolge, welche mit einem Backslash `\` beginnt, wird auch „**Ersatzdarstellung**“ (auf Englisch „**escape sequence**“) genannt, was in Kapitel → 6.5.5 genauer besprochen werden wird.

Die Zeichenfolge `%d` veranlasst die Konsole, eine Dezimalzahl (`d`) auszugeben. Die gewünschte Zahl wird als zusätzliches Argument an die Funktion `printf()` in den runden Klammern angegeben: `zahl * zahl`. Mit Hilfe des Aufrufes der Funktion `printf()` wird somit die Zahl ausgegeben, die der Quadratzahl der Variablen `zahl` während des aktuellen Schleifendurchlaufs entspricht.

> Beachten Sie die Reihenfolge: `printf()` muss als erstes Argument den Format-String und dann anschließend die anderen auszugebenden Ausdrücke erhalten. Für jedes Formatelement im Format-String muss eins-zu-eins ein entsprechender Ausdruck existieren. Dabei müssen die Formatelemente und die auszugebenden Ausdrücke im Typ übereinstimmen.

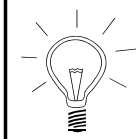


Weitere Beispiele und Erklärungen zu der Funktion `printf()` und den Formatelementen sind in den folgenden Beispielprogrammen zu finden. Eine ausführliche Beschreibung findet sich in Kapitel → 16.7.

2.1.7 Inkludieren von Bibliotheksfunktionen

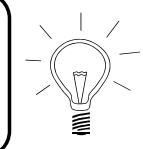
Die Sprache C selbst besitzt keine eingebauten Funktionen für die Ein- und Ausgabe am Bildschirm. In C werden hierfür Funktionen mit standardisierten Schnittstellen in sogenannten Standardbibliotheken bereitgestellt. Die Schnittstellen der Funktionen stehen in sogenannten „**Header-Dateien**“ (auf Englisch **header files** oder kurz **h-files**). Durch Einbinden der entsprechenden Header-Datei in das eigene Programm ist es möglich, die **Ein- und Ausgabefunktionen** der Bibliotheken zu nutzen.

> Mit Hilfe der **`#include`**-Anweisung ist es möglich, eine externe Datei für die Dauer des Übersetzungslaufs in den Quellcode zu kopieren. Eine Header-Datei enthält alle Angaben zu Funktionen, Typen und externen Variablen, welche in einer Bibliothek enthalten sind.



Der C Standard umfasst mehrere Standard-Bibliotheken wie hier im Beispiel `stdio`, in welcher sich unter anderem die Schnittstelle der Funktion `printf()` befindet. Erst mithilfe der passenden `#include`-Anweisung wird somit ein Aufruf der Funktion `printf()` überhaupt erst möglich.

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



2.2 Celsius und Fahrenheit

Folgendes Programm erzeugt eine Temperaturtabelle zur Umrechnung von der Einheit Fahrenheit in die Einheit Celsius:

fahrenheit.c

```
#include <stdio.h>

// Konstanten
#define UPPER      200
#define LOWER        0
#define STEP_SIZE   20

/* Diese Funktion gibt eine Tabelle aus,
 * in welcher Fahrenheit-Werte in die passenden
 * Celsius-Werte umgerechnet wurden.
 */
int main(void) {

  // Schleife von LOWER bis UPPER
  for (int fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP_SIZE) {
    int celsius = 5 * (fahr - 32) / 9;
    printf("%3d\t%3d\n", fahr, celsius);
  }

  return 0;  // Ende des Programmes
}
```

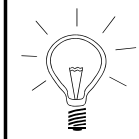
Dieses Programm gibt aus:

```
  0     -17
 20      -6
 40       4
 60      15
 80      26
100      37
120      48
140      60
160      71
180      82
200      93
```

2.2.1 Kommentare

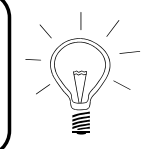
Das erste, was in diesem Programm auffällt, sind die neu hinzugefügten Kommentare.

Ein **Kommentar** dient zur Dokumentation eines Programms und hat keinen Einfluss auf dessen Ablauf. Kommentare sollten immer dann geschrieben werden, wenn ein Programm nicht selbsterklärend ist.

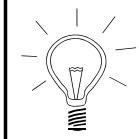


In C gibt es zwei Varianten von Kommentaren: Ein einzeiliger Kommentar und ein Kommentarblock.

Ein **einzeiliger Kommentar** wird durch zwei Slash-Zeichen `//` eingeführt. Alles, was in derselben Zeile hinter diesen beiden Zeichen steht, wird vom Compiler ignoriert.



Alles, was zwischen den Zeichenfolgen `/*` und `*/` steht, stellt einen sogenannten **„Kommentarblock“** dar. Kommentarblöcke können über mehrere Zeilen gehen, dürfen aber nicht in einander verschachtelt werden.

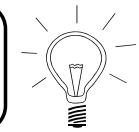


Der einzeilige Kommentar wird heute in vielen Programmiersprachen bevorzugt und wird auch durch Entwicklungsumgebungen standardmäßig unterstützt. Tatsächlich war aber in C ursprünglich nur der Kommentarblock möglich. Erst seit dem Standard C99 sind einzeilige Kommentare erlaubt.

2.2.2 Makros

Die Größen `LOWER`, `UPPER` und `STEP_SIZE` sind sogenannte **Makros** und werden auch „**symbolische Konstanten**“ genannt.

> Makros werden üblicherweise mit Großbuchstaben und dem Underscore-Zeichen als Trenner geschrieben.



Mithilfe der Anweisung **`#define`** kann definiert werden, was ein bestimmter Name innerhalb des Codes representieren soll. Dies bedeutet, dass diese Namen keine Variablen darstellen, sondern einfach nur vom Compiler durch den Wert ersetzt werden, mit welchem sie definiert wurden.

Diese Ersetzung übernimmt der sogenannte „**Präprozessor**“ (auf Englisch „**preprocessor**“). Dieser bearbeitet die Dateien vor dem eigentlichen Compilieren. Anweisungen an den Präprozessor beginnen stets mit einem Nummernzeichen #. Somit ist auch die `#include` Anweisung eine Präprozessor-Anweisung.

Mehr dazu und auch wieso diese Konstanten „Makros“ genannt werden, kann im Kapitel → 21 nachgelesen werden.

2.2.3 `for`-Schleife

In diesem Beispiel wurde keine `while`-Schleife verwendet, sondern eine `for`-Schleife. Eine `for`-Schleife eignet sich besonders gut für **Zählschleifen**. Hierbei wird stets eine Zählvariable (auch „**Schleifenvariable**“ genannt) von einem Initialwert bis zu einem Zielwert gezählt.

```
for (int fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
```

Die `for`-Schleife definiert drei Elemente, jeweils getrennt durch ein Semikolon:

- Initialisierung der Zählvariable.
- Bedingung, unter welcher die Schleife ausgeführt wird.
- Schritt, mit welchem die Zählvariable verändert wird.

In der for-Schleife stellt somit `fahr = LOWER` die Initialisierung dar, `fahr <= UPPER` die **Bedingung**, wie lange die Schleife durchgeführt wird, und `fahr = fahr + STEP` berechnet den nächsten Wert, für den die Schleife durchgeführt wird.

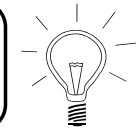
Vergleicht man dieses Beispiel mit dem Beispiel aus Kapitel → 2.1, so wird klar, dass auch dort bereits eine for-Schleife hätte verwendet werden können.

Die verschiedenen Schleifen werden in Kapitel → 10 behandelt.

2.2.4 Formatierung von Dezimalzahlen bei `printf()`

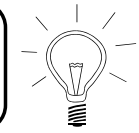
Für die Formatierung der Zahlen wurden bei printf() zwei neue Formatelemente verwendet: `\t` und `%3d`.

> Mit der Zeichenfolge `\t` wird auf der Konsole ein Tabulator-Schritt ausgegeben.



Durch die Verwendung des Tabulator-Zeichens als Trennzeichen zwischen den Werten werden die Celsius-Werte auf der Konsole mit einem größeren Abstand ausgegeben. Viele Programme erkennen solche Tabulatoren und ordnen die Werte dann automatisch in Spalten an.

> Mit der Zeichenfolge `%3d` wird eine Dezimalzahl rechtsbündig mit 3 Ziffern Platz ausgegeben.



Eine ausführliche Beschreibung aller Formatelemente kann in Kapitel → 16.7 nachgelesen werden.

2.2.5 Ausdrücke

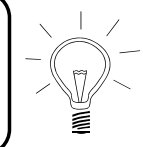
Die Berechnung der Temperatur liest sich wie ein mathematischer **Ausdruck**:

```
int celsius = 5 * (fahr - 32) / 9;
```

Genau darauf baut die Sprache C auf: Wie in der Mathematik können Operanden wie Variablen und Zahlen mit Operatoren wie Addition, Subtraktion, Multiplikation und Division verknüpft werden. Genauso wie in der Mathematik werden auch die runden Klammern verwendet, um anzuzeigen, welche Teil-Ausdrücke priorisiert behandelt werden sollen.

Die Länge eines Ausdrucks ist unbegrenzt, es ist jedoch üblich, Ausdrücke so einfach wie möglich zu gestalten und gegebenfalls Zwischenresultate wiederum in Variablen zwischenzuspeichern.

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



2.2.6 `int`-Zahlen und `double`-Zahlen

In dem Beispiel wurde bislang stets mit Integern (Ganzzahlen), sprich dem Typ `int` gerechnet. Dies funktionierte bislang ganz gut, birgt jedoch einige Gefahren. Man könnte den Ausdruck zur Berechnung des Celsius-Wertes beispielsweise auch so schreiben:

```
int celsius = 5 / 9 * (fahr - 32);
```

Rein mathematisch würde dies dasselbe Resultat ergeben. In der Programmierung jedoch entsteht dadurch ein Problem: Die beiden Zahlen `5` und `9` sind Integer, weswegen der Compiler automatisch eine sogenannte „Integer-Division“ (eine Division ohne Rest) durchführt. Diese Division `5/9` ergibt `0`. Das Resultat sämtlicher Werte wird somit null sein.

Um dieses Problem zu lösen, muss der Ausdruck mit sogenannten „**Gleitpunkt-Typen**“ (Zahlen mit Nachkommastellen) geschrieben werden:

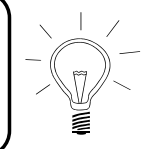
```
double celsius = 5. / 9. * (fahr - 32.);
```

Als erstes wird die Variable `celsius` nicht mehr als `int` definiert, sondern als `double`. Dies ist in C der Standard-Typ für Gleitpunkt-Typen. Die Variable `celsius` kann somit auch Zahlen mit Nachkommastellen speichern.

Des Weiteren wurden die Zahlen des Ausdrucks mit einem Punkt versehen. Damit wird der Compiler angewiesen, diese Zahlen mit dem Typ `double` zu verarbeiten. Entsprechend wird keine Integer-Division mehr ausgeführt, sondern eine normale Division mit Nachkommastellen.

Die Konvertierung der `int`-Variablen `fahr` in eine `double`-Variable erfolgt hier implizit (siehe auch Kapitel → 9.10).

> Man spricht von einer **impliziten Typumwandlung**, wenn eine Konvertierung vorgenommen wird, ohne dass dies explizit in der Programmiersprache formuliert werden muss.



Die Ausgabe einer Zahl vom Typ `double` auf der Konsole benötigt ein neues Formatelement: `%f`

```
printf("%3d\t%f\n", fahr, celsius);
```

Fügt man diese beiden Änderungen in das Beispiel ein, so ergibt sich folgende Ausgabe:

```
  0     -17.777778
 20     -6.666667
 40     4.444444
 60     15.555556
 80     26.666667
100     37.777778
120     48.888889
140     60.000000
160     71.111111
180     82.222222
200     93.333333
```

2.3 Zinsberechnung

Das folgende Beispiel berechnet die jährliche Entwicklung eines Grundkapitals über eine vorgegebene Laufzeit. Die Bank gewährt zwei unterschiedliche Zinsen über zwei Laufzeiten. Die Zinsen sollen jeweils nicht ausgeschüttet, sondern mit dem Kapital wieder angelegt werden. Es wird eine Tabelle mit folgenden Angaben erzeugt: Laufendes Jahr und angesammeltes Kapital (in EUR).

Gegeben seien das Grundkapital (1000 EUR), der Zins (5%) über die erste Laufzeit (3 Jahre) und der Zins (2.5%) über die zweite Laufzeit (5 Jahre).

zins.c

```
#include <stdio.h>

#define ZINS_1       5.0
#define ZINS_2       2.5
#define LAUFZEIT_1   3
#define LAUFZEIT_2   5
#define GRUNDKAPITAL 1000.00

void printKapital(int jahr, double kapital) {
  printf("Jahr: %2d\t", jahr);
  printf("Kapital: %7.2f EUR\n", kapital);
}

int main(void) {
  double kapital = GRUNDKAPITAL;
  printKapital(0, kapital);

  int totalLaufzeit = LAUFZEIT_1 + LAUFZEIT_2;
  for (int jahr = 1; jahr <= totalLaufzeit; jahr = jahr + 1) {
    if (jahr <= LAUFZEIT_1) {
      kapital = kapital * (1. + ZINS_1 / 100.);
    } else {
      kapital = kapital * (1. + ZINS_2 / 100.);
    }
    printKapital(jahr, kapital);
  }

  printf("\n");
  printf("Aus %7.2f EUR Grundkapital\n", GRUNDKAPITAL);
  printf("wurden in %d Jahren %7.2f EUR\n", totalLaufzeit, kapital);
  return 0;
}
```

Genauso wie in den vorangegangenen Beispielen können auch hier wieder folgenden Punkte beobachtet werden:

- Grundkapital, Laufzeiten und Zinssätze werden als konstante Größen angesehen. Entsprechend wurden sie als symbolische Konstanten (Makros) definiert.
- Die Variable `kapital` ist eine Variable vom Datentyp `double`. Dies ist notwendig, um die Zinsberechnung mit korrekten Nachkommastellen durchzuführen. Die Zählvariable `jahr` hingegen ist vom Datentyp `int`.
- Um Schreibarbeit zu sparen, wurde die Gesamtlaufzeit einmal berechnet und in der Variable `totalLaufzeit` gespeichert.
- In der `for`-Schleife stellt der Initialisierungsausdruck `jahr = 1` den Beginn der Schleife dar. `jahr <= totalLaufzeit` ist die Bedingung, wie lange die Schleife durchzuführen ist, und `jahr = jahr + 1` berechnet den nächsten Wert, für den die Schleife durchgeführt wird.
- Die Funktion `printf()` wird wiederum benutzt, um Text und Werte formatiert auszugeben. Mit dem Steuerzeichen `'\n'` wird der Cursor an den Beginn der nächsten Bildschirmzeile positioniert und mit `'\t'` ein Tabulatorschritt eingefügt.
- Neu hinzugekommen ist in diesem Beispiel das Formatelement `%7.2f`, was bedeutet, dass die Zahl mit insgesamt 7 Zeichen formatiert werden soll. Hierbei sollen 2 Ziffern für die Nachkommastellen benötigt werden. Ein Zeichen wird für den Dezimalpunkt benötigt, somit bleiben noch 4 Ziffern für die Zahl vor dem Dezimalpunkt übrig.

Das Programm gibt aus:

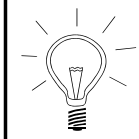
```
Jahr:  0     Kapital: 1000.00 EUR
Jahr:  1     Kapital: 1050.00 EUR
Jahr:  2     Kapital: 1102.50 EUR
Jahr:  3     Kapital: 1157.62 EUR
Jahr:  4     Kapital: 1186.57 EUR
Jahr:  5     Kapital: 1216.23 EUR
Jahr:  6     Kapital: 1246.64 EUR
Jahr:  7     Kapital: 1277.80 EUR
Jahr:  8     Kapital: 1309.75 EUR

Aus 1000.00 EUR Grundkapital
wurden in 8 Jahren 1309.75 EUR
```

2.3.1 `if`-Struktur

Ein sehr wichtiges Element der Programmierung ist die `if`-Struktur.

> Die `if`-Struktur wird auch als **einfache Selektion** bezeichnet. Es werden zwei Alternativen ausprogrammiert, wobei die Entscheidung, welche der beiden Alternativen schlussendlich durchgeführt wird, auf einer Bedingung basiert, welche während des Programmablaufes ausgewertet wird.



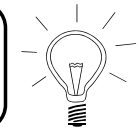
Hier werden mit der `if`-Struktur die beiden Laufzeiten unterschieden. Falls das aktuelle Jahr kleiner oder gleich ist wie `LAUFZEIT_1`, so wird der Zinssatz `ZINS_1` für die Berechnung verwendet, ansonsten der Zinssatz `ZINS_2`.

Die `if`-Struktur wird im Detail in Kapitel → 10.2 behandelt.

2.3.2 Funktionsaufruf

In diesem Beispiel wurde ein weiteres, wichtiges Element benutzt: Eine **Funktion**.

> Funktionen verkapseln eine Funktionalität und können an beliebigen Stellen aufgerufen werden.



Hier wurde die Funktion `printKapital()` definiert, welche eine einzige Zeile der Tabelle schön formatiert ausgibt. Um dies zu tun, erwartet die Funktion zwei sogenannte „**Parameter**“: Das Jahr und das Kapital. Innerhalb dieser Funktion können diese beiden Angaben als Variablen angesprochen werden.

Um die Funktion aufzurufen, wird einfach der Funktionsname geschrieben und innerhalb der runden Klammern werden die Werte angegeben, welche als sogenannte „**Argumente**“ an die Funktion übergeben werden sollen.

Diese Argumente können beliebige Ausdrücke sein. Insbesondere sind Variablen erlaubt wie `jahr` und `kapital` aber auch feste Werte wie `0` oder symbolische Konstanten wie `GRUNDKAPITAL`.

Auch die `printf()`-Funktion ist eine Funktion. Entsprechend können auch dort beliebige Ausdrücke übergeben werden. Im Beispiel → 2.1 wurde der Ausdruck `zahl * zahl` übergeben.

Das Schlüsselwort `void` bedeutet, dass diese Funktion keinen Wert zurückgibt. Diese und andere Eigenschaften von Funktionen werden im Detail in Kapitel → 4.3 behandelt. Hier an dieser Stelle genügt es, einmal gesehen zu haben, wie eine Funktion geschrieben und aufgerufen wird.