ANSI C

Grundlagen der Programmierung

CANSI2

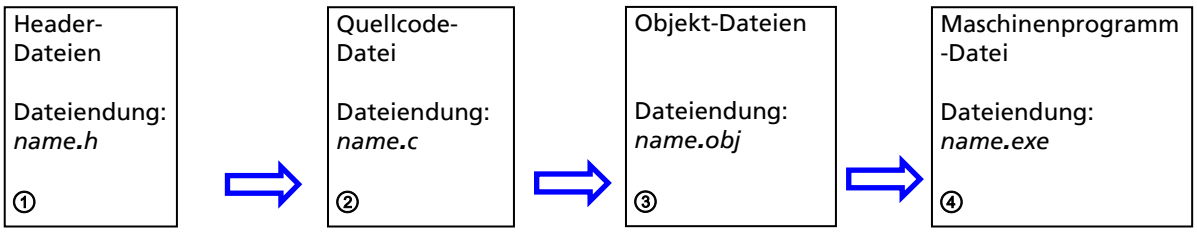
Ricardo Hernández García

3. Ausgabe, September 2011

HERDT

2.4 Vom Quellcode zum Maschinenprogramm

Das folgende Beispiel geht von der Erstellung einer C-Anwendung aus, die unter dem Betriebssystem Windows laufen soll, was an der Dateiendung **.exe* der fertigen Anwendung zu erkennen ist. Der Vorgang wäre allerdings unter Linux völlig identisch.



Folgende Schritte werden durchlaufen, bevor aus einem C-Programm ein Maschinenprogramm erstellt wird:

- ✔ Zuerst werden die Header-Dateien durch den Präcompiler in die Quellcodedatei eingefügt. Dieses Verfahren ermöglicht, Standardbibliotheken und fremde Funktionen in das Programm einzubinden, ohne dass die Programmteile vom Programmierer in den Quelltext kopiert werden müssen. Die Header-Datei liefert damit Informationen zu einer Funktionalität, die an anderer Stelle bereitgestellt wird. Der Präcompiler wird über sogenannte Direktiven gesteuert. Das sind Befehle, die im Quellcode mit dem Zeichen # beginnen.
- ✔ Im folgenden Schritt wird mit dem Compiler der Quellcode in eine Objektdatei übersetzt (diese hat meist die Endung *.o* oder **.obj*). Objektdateien sind Maschinenprogramme, deren Lage im Speicher des Computers noch durch den Linker angepasst werden müssen.
- ✔ Im letzten Schritt werden gegebenenfalls noch weitere Objektdateien mit dem Hauptprogramm verbunden (z. B. Bibliotheken von Drittherstellern). Dieser Vorgang wird "**Linken**" genannt. Das Programm ist jetzt vom Quellcode in ein ausführbares Maschinenprogramm übersetzt und gelinkt worden und kann direkt vom Prozessor abgearbeitet werden.

Präcompiler

Der Präcompiler (auch Präprozessor genannt) startet seine Aktivitäten, bevor der eigentliche Compiler den Quelltext übersetzt. Seine Aufgabe besteht darin, die Header-Dateien in den Quelltext einzufügen, weitere Definitionen bereitzustellen und Vorbereitungen für den Compiler zu treffen, damit eine Übersetzung erfolgreich durchgeführt werden kann.

Anweisungen, die im Quelltext für den Präcompiler gedacht sind, werden mit dem Zeichen # am Anfang der Zeile versehen. Die folgende Anweisung bindet zum Beispiel die Header-Datei `stdio.h` ein, die die Standardfunktionen für die Ein- und Ausgabe bereitstellt.

```
#include <stdio.h>
```

Ohne diese Anweisung ist z. B. die Funktion `printf`, die für Bildschirmausgaben benötigt wird, dem Compiler unbekannt und er gibt eine entsprechende Fehlermeldung zurück.

Eine detaillierte Übersicht über den Präcompiler und die zahlreichen Präcompiler-Direktiven finden Sie in Kapitel 11.

C-Compiler und Entwicklungsumgebungen

Damit Sie C-Programme schreiben, übersetzen und ausführen können, benötigen Sie einen C-Compiler. Dazu gibt es eine sehr große Auswahl. Zumeist sind diese Umgebungen für die Sprache C++ vorgesehen. Sie unterstützen aber auch die Programmerstellung in C. Es folgt eine Übersicht über häufig eingesetzte Compiler/Entwicklungsumgebungen.

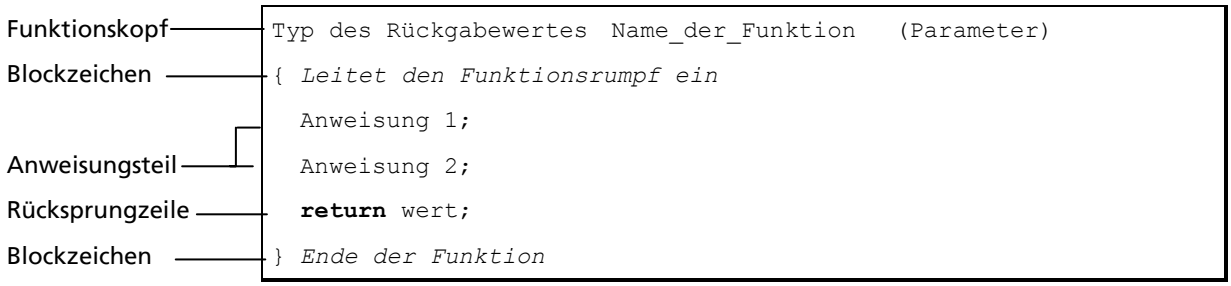
Übersicht über C-Compiler

Compiler	Kurzbeschreibung
GNU Compiler	GCC (GNU Compiler Collection) ist eine Sammlung verschiedener Compiler, unter anderem auch für C/C++. Sie besitzt im einfachsten Fall einen Kommandozeilen-Compiler, den Sie manuell in einer Konsole aufrufen können. Viele Entwicklungsumgebungen verwenden Compiler aus dieser Sammlung. Als Zielplattform stehen Windows sowie zahlreiche Linux- und UNIX-Versionen zur Auswahl. Der Compiler und häufig auch die Entwicklungsumgebungen sind kostenfrei. *http://gcc.gnu.org/*
C++ Builder	Hier handelt es sich um eine Weiterentwicklung von Borland C++, der von Embarcadero weiterentwickelt wird. Aktuell stehen nur kommerzielle Versionen zur Verfügung. Die Entwicklung erfolgt nur für Windows. *http://www.embarcadero.com/de/products/cbuilder*
Visual Studio bzw. Visual Studio Express C++	Microsoft stellt verschiedene kommerzielle wie auch kostenfreie Versionen bereit. Die fertigen Anwendungen laufen nur unter Windows. *http://www.microsoft.com/germany/visualstudio/* *http://www.microsoft.com/germany/express/*
Eclipse CDT	Auf Basis der Eclipse-Plattform wird hier eine Entwicklungsumgebung für C/C++ bereitgestellt, die als Compiler auf GNU zurückgreift. Damit stehen die Entwicklungsumgebung sowie die fertigen Anwendungen auf unterschiedlichsten Plattformen zur Verfügung. Die Software ist kostenfrei. *http://www.eclipse.org/cdt/*

Eine umfangreiche Liste mit verfügbaren C/C++-Compilern finden Sie unter *http://www.thefreecountry.com/compilers/cpp.shtml*.

5.2 Definition einer Funktion

Aufbau einer Funktion



Syntax der Funktionsdefinition

```
int add_mwst(int z1, int z2)
{
  return z1 + z2;
}
```

- ✔ Nach den Präcompiler-Anweisungen wird der Funktionskopf geschrieben. Er enthält an erster Stelle den Rückgabetyp der Funktion. Der Rückgabetyp der Funktion muss mit dem Typ des Rückgabewertes (Wert nach der Anweisung `return`) übereinstimmen.
- ✔ An zweiter Stelle wird der Name der Funktion, der sich an die Vorgaben von Bezeichnern hält, angegeben.
- ✔ In runden Klammern eingeschlossen folgt die Parameterliste. Über die Parameter der Liste werden Werte an die Funktion übergeben. Mehrere formale Parameter werden durch Kommata getrennt.
- ✔ Eine Parameterangabe besteht immer aus der Angabe eines Datentyps und eines Bezeichners. Der Bezeichner kann dann in der Funktion wie eine Variable verwendet werden. Für einen Parameter wird in der Funktion eine Kopie des Wertes angelegt, sodass sich dessen Änderung in der Funktion nicht auf den originalen Wert der übergebenen Variablen im Hauptprogramm auswirkt.
- ✔ Es folgt der Anweisungsblock der Funktion. Anweisungsblöcke werden von den Zeichen (geschweifte Klammern) { und } eingeschlossen.
- ✔ Innerhalb des Anweisungsteils erfolgt der Rücksprung zur funktionsaufrufenden Anweisung. Der Wert nach der Anweisung `return` wird als Rückgabewert der Funktion zurückgegeben. Über diesen Wert können Sie z. B. den Erfolg der Ausführung der Funktion oder ein Berechnungsergebnis zurückgeben.

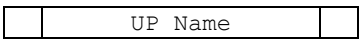
Verwenden einer Funktion

- ✔ Der Funktionsaufruf erfolgt durch den Funktionsnamen und eine Liste mit Variablen, Konstanten oder Ausdrücken, deren Werte an die Funktion übergeben werden sollen.

```
Funktionsname(Parameter1, Parameter2, ...);      /* Aufruf ohne Rueckgabewert
*/
Var = Funktionsname(Parameter1, Parameter2, ...);/* Aufruf mit Wertrueckgabe */
```

- ✔ Die Anzahl der übergebenen Parameter muss mit denen der Funktionsdefinition übereinstimmen, ebenso der Datentyp.
- ✔ Gibt die Funktion einen Wert zurück, wird sie wie ein Ausdruck einer Variablen zugewiesen oder in einem Ausdruck verwendet.
- ✔ Wenn eine Funktion einen Wert zurückgibt, muss mindestens eine `return`-Anweisung in der Funktion vorhanden sein. Die Klammern sind optional und nicht unbedingt notwendig. Der Rückgabewert kann auch über einen Ausdruck angegeben werden, der den entsprechenden Typ liefert.
- ✔ Es können in einer Funktion mehrere `return`-Anweisungen angegeben werden. Dadurch sind verschiedene Positionen zum Verlassen der Funktion möglich.
- ✔ Um eine Funktion verwenden zu können, muss sie bereits deklariert worden sein.

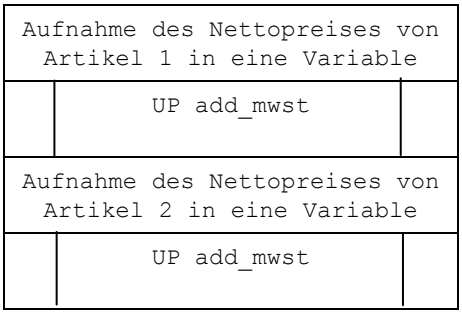
Struktogramm-Symbol



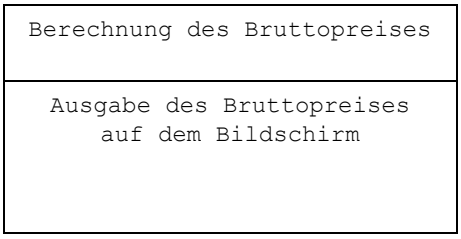
Das Struktogramm-Symbol unterscheidet sich nur wenig von dem einer Sequenz. Es bekommt links und rechts vom rechteckigen Rahmen zwei senkrechte Linien. Diese weisen auf einen Unterprogrammaufruf (Funktionsaufruf) hin. Das Struktogramm für das Unterprogramm wird in einem separaten Struktogramm gezeichnet.

Struktogramm des Hauptprogramms und des Unterprogramms (Funktion)

Hauptprogramm

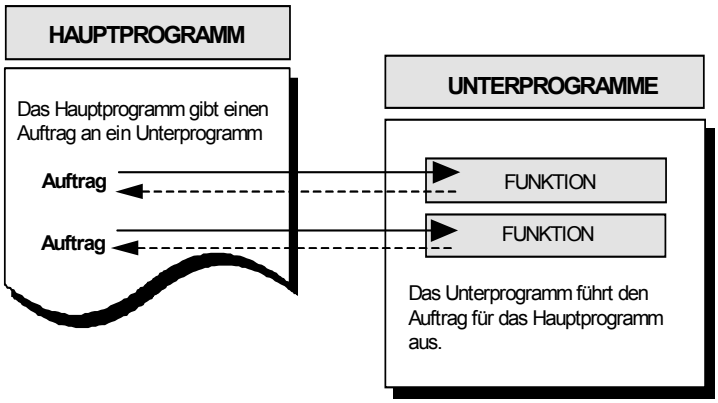


Unterprogramm



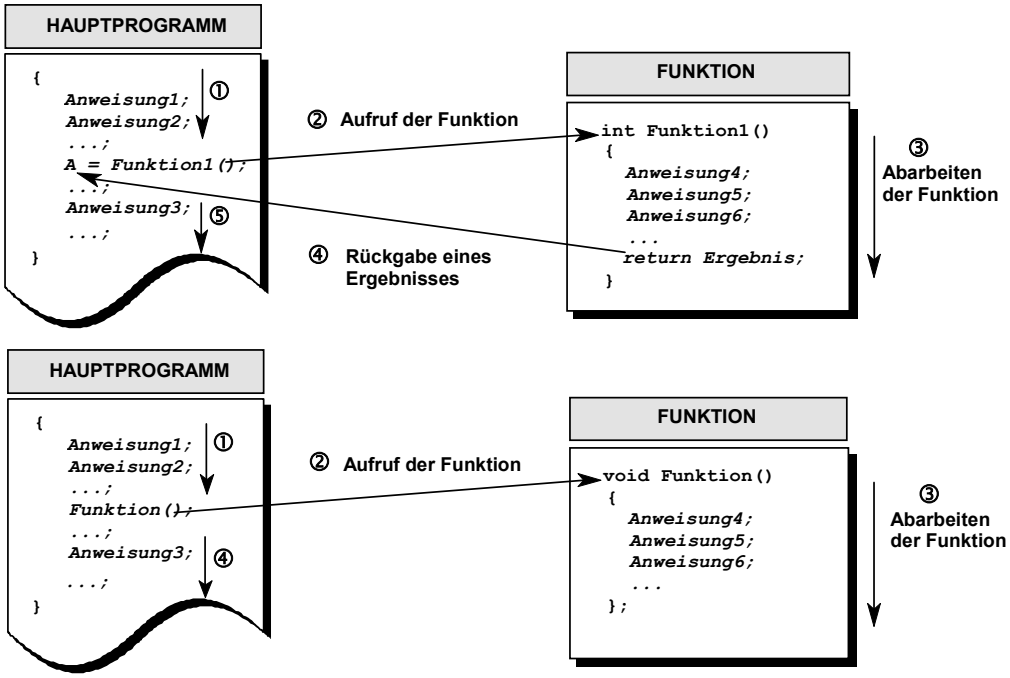
Unterprogrammaufruf als Auftrag der Hauptfunktion

Das Unterprogramm beinhaltet einen Algorithmus, der mehrfach in der Programmausführung benötigt wird. Durch das Auslagern eines Algorithmus zur Lösung einer Teilaufgabe in ein Unterprogramm muss der Algorithmus nur einmal programmiert und getestet werden, kann aber mehrmals mit verschiedenen Argumenten aufgerufen werden. Auf diese Weise erhalten Sie einen übersichtlichen und kompakten Quellcode. Über die Unterprogrammtechnik ist es möglich, ein kompliziertes Gesamtproblem in kleinere Teilprobleme zu zerlegen, die miteinander kombiniert die Lösung des Gesamtproblems ergeben. Außerdem ist dann die Wiederverwendung fertiger, getesteter Funktionen möglich.



Funktion mit und ohne Ergebnisrückgabe

Das Unterprogramm kann ein Ergebnis zurückgeben. Dazu werden die Sequenzen des Unterprogramms abgearbeitet und mit der Anweisung `return` wird der Rückgabewert an die rufende Sequenz übergeben. Funktionen, die kein Ergebnis zurückgeben, werden auch als `void`-Funktionen bezeichnet.



Beispiel: *NettoBrutto1.c*

Im folgenden Programm werden die Artikelpreise und der Mehrwertsteuersatz über Konstanten vereinbart. Die Funktion `add_mwst` übernimmt die Berechnung der Mehrwertsteuer und addiert den Wert zum Artikelpreis. Der berechnete Artikel-Bruttopreis wird auf dem Bildschirm ausgegeben.

```
①  #include <stdio.h>
②  const double mwst_satz = 0.19;
   const double artikel_1 = 7.20;
   const double artikel_2 = 1.40;
   const double artikel_3 = 5.60;
③  double preis;
④  void add_mwst(void)
   {
⑤    printf("\n+\t%f  EUR\n", preis * (mwst_satz + 1));

   }
⑥  int main(void)
   {
⑦    preis = artikel_1;
⑧    add_mwst();
⑨    preis = artikel_2;
     add_mwst();
     preis = artikel_3;
     add_mwst();
     printf("-----------------");
     preis = artikel_1 + artikel_2 + artikel_3;
     add_mwst();
     printf("=================");
     getchar();
     return 0;
   }
```

① Die Header-Datei *stdio.h* wird eingebunden, um die Funktion `printf` verwenden zu können.
② Der Mehrwertsteuersatz wird in einer Konstanten vereinbart.
③ In dieser Programmzeile wird die Variable `preis` vom Typ `double` (Gleitkommazahl) definiert.
④ Der Kopf der Funktion `add_mwst` wird in dieser Zeile vereinbart. Die Funktion hat keine Parameter (`void`) und gibt `void` zurück.
⑤ In dieser Zeile wird der Nettopreis mit dem Mehrwertsteuersatz + 1 (Faktor 1,19) multipliziert und das Ergebnis auf dem Bildschirm ausgegeben.
⑥ Hier beginnt das Hauptprogramm.
⑦ In dieser Zeile erfolgt die Zuweisung des Artikelpreises zur Variablen `preis`.
⑧ Die Funktion `add_mwst` wird aufgerufen. Dadurch wird der Bruttopreis berechnet und ausgegeben.
⑨ Der nächste Artikelpreis wird der Variablen `preis` zugewiesen und die Abarbeitung analog nochmals mit den eben beschriebenen Schritten fortgesetzt.

5.3 Parameter einer Funktion

Über die im Funktionskopf vereinbarten Parameter können Sie Werte über Variablen oder Konstanten an eine Funktion übergeben. Innerhalb der Funktion können Sie über deren Bezeichner auf die Parameterwerte zugreifen.

Beispiel: *NettoBrutto2.c*

Im folgenden Beispiel wird über die Funktion `add_mwst` die Mehrwertsteuer des im Parameter übergebenen Preises eines Artikels auf dem Bildschirm ausgegeben. Dazu wird die Funktion `add_mwst` in der Funktion `main` mit jedem Artikel einzeln und anschließend mit der Summe der Artikel aufgerufen.

```
   #include <stdio.h>
   const double mwst_satz = 0.19;
   const double artikel_1 = 5.20;
   const double artikel_2 = 1.25;
   const double artikel_3 = 0.50;
①  void add_mwst(double preis)
   {
②    printf("\n*\t%5.2f  EUR\n", preis * (mwst_satz + 1));}
③  int main()
   {
④    add_mwst(artikel_1);
     add_mwst(artikel_2);
     add_mwst(artikel_3);
⑤    printf("---------------------\n");
⑥    add_mwst(artikel_1 + artikel_2 + artikel_3);
⑦    printf("=====================\n\n");
     getchar();
     return 0;
   }
```

① Hier steht der Kopf der Funktion `add_mwst` mit dem Parameter `preis` vom Typ `double`.
② Auf dem Bildschirm werden ein Zeilenumbruch, das Zeichen * und anschließend ein Tabulator, gefolgt vom Wert der Variablen, ausgegeben. Der Wert der Variablen wird 5-stellig mit 2 Stellen nach dem Komma formatiert. Weiterhin erfolgt die Ausgabe des Währungssymbols *EUR* und ein Zeilenumbruch. Der auszugebende Wert ist das Produkt aus dem Preis und der Summe des Mehrwertsteuersatzes von `0.19` und `1` (`1.19`).
③ In dieser Zeile beginnt das Hauptprogramm über die Funktion `main`.

① Die Header-Datei *stdio.h* wird eingebunden, um die Funktion `printf` verwenden zu können.
② Der Mehrwertsteuersatz wird in einer Konstanten vereinbart.
③ In dieser Programmzeile wird die Variable `preis` vom Typ `double` (Gleitkommazahl) definiert.
④ Der Kopf der Funktion `add_mwst` wird in dieser Zeile vereinbart. Die Funktion hat keine Parameter (`void`) und gibt `void` zurück.
⑤ In dieser Zeile wird der Nettopreis mit dem Mehrwertsteuersatz + 1 (Faktor 1,19) multipliziert und das Ergebnis auf dem Bildschirm ausgegeben.
⑥ Hier beginnt das Hauptprogramm.
⑦ In dieser Zeile erfolgt die Zuweisung des Artikelpreises zur Variablen `preis`.
⑧ Die Funktion `add_mwst` wird aufgerufen. Dadurch wird der Bruttopreis berechnet und ausgegeben.
⑨ Der nächste Artikelpreis wird der Variablen `preis` zugewiesen und die Abarbeitung analog nochmals mit den eben beschriebenen Schritten fortgesetzt.

5.3 Parameter einer Funktion

Über die im Funktionskopf vereinbarten Parameter können Sie Werte über Variablen oder Konstanten an eine Funktion übergeben. Innerhalb der Funktion können Sie über deren Bezeichner auf die Parameterwerte zugreifen.

Beispiel: *NettoBrutto2.c*

Im folgenden Beispiel wird über die Funktion `add_mwst` die Mehrwertsteuer des im Parameter übergebenen Preises eines Artikels auf dem Bildschirm ausgegeben. Dazu wird die Funktion `add_mwst` in der Funktion `main` mit jedem Artikel einzeln und anschließend mit der Summe der Artikel aufgerufen.

```
   #include <stdio.h>
   const double mwst_satz = 0.19;
   const double artikel_1 = 5.20;
   const double artikel_2 = 1.25;
   const double artikel_3 = 0.50;
①  void add_mwst(double preis)
   {
②    printf("\n*\t%5.2f  EUR\n", preis * (mwst_satz + 1));}
③  int main()
   {
④    add_mwst(artikel_1);
     add_mwst(artikel_2);
     add_mwst(artikel_3);
⑤    printf("---------------------\n");
⑥    add_mwst(artikel_1 + artikel_2 + artikel_3);
⑦    printf("=====================\n\n");
     getchar();
     return 0;
   }
```

① Hier steht der Kopf der Funktion `add_mwst` mit dem Parameter `preis` vom Typ `double`.
② Auf dem Bildschirm werden ein Zeilenumbruch, das Zeichen * und anschließend ein Tabulator, gefolgt vom Wert der Variablen, ausgegeben. Der Wert der Variablen wird 5-stellig mit 2 Stellen nach dem Komma formatiert. Weiterhin erfolgt die Ausgabe des Währungssymbols *EUR* und ein Zeilenumbruch. Der auszugebende Wert ist das Produkt aus dem Preis und der Summe des Mehrwertsteuersatzes von `0.19` und `1` (`1.19`).
③ In dieser Zeile beginnt das Hauptprogramm über die Funktion `main`.

④ Die Funktion `add_mwst` wird aufgerufen und der Wert der Konstanten `artikel_1` als Parameter übergeben.

⑤ In dieser Zeile wird eine gestrichelte Linie zur Trennung der Einzelpreise und der Summe auf dem Bildschirm ausgegeben.

⑥ Hier erfolgt der Aufruf der Funktion `add_mwst`, vgl. Punkt ④. Im Aufruf wird zunächst eine Summe berechnet und das Ergebnis der Funktion als Parameter übergeben.

⑦ In Anlehnung an die Notation der Mathematik wird eine gestrichelte Doppellinie zur Hervorhebung des Ergebnisses auf dem Bildschirm ausgegeben.

5.4 Rückgabe von Werten

Funktionen geben Ihren Rückgabewert über die `return`-Anweisung an den Aufrufer zurück. Die Funktion wird nach dem Aufruf sofort verlassen. Wird kein Wert zurückgegeben, kann auf die Angabe von `return` verzichtet werden.

Beispiel: *EurUsd.c*

Im Beispiel *EurUsd.c* wird der Artikelpreis in EUR übergeben und nach der Umrechnung (hier zu einem Beispielwechselkurs von 1.43372) wird der Preis in US-Dollar zurückgegeben.

```
   #include <stdio.h>
   const double usd_kurs = 1.43372;
   const double artikelpreis = 23.45;
①  double ergebnis;
②  double eur_usd(double preis)
   {
③    return preis * usd_kurs;
   }
④  int main(void)
   {
⑤    ergebnis = eur_usd(artikelpreis);
     printf("Der Artikelpreis betraegt %5.2f EUR \n das entspricht %5.2f USD.\n",
     artikelpreis, ergebnis);
     printf("------ oder durch den direkten Funktionsaufruf -----\n");
⑥    printf("Der Artikelpreis betraegt %5.2f EUR \n das entspricht %5.2f USD.\n",
     artikelpreis, eur_usd(artikelpreis));
     getchar();
     return 0;
   }
```

① Es wird die Variable `ergebnis` vom Typ `double` (Gleitkommazahl) vereinbart.

② Hier steht der Funktionskopf der Funktion `eur_usd`. Die Funktion berechnet aus einer Euro-Angabe einen equivalenten Dollarwert unter Berücksichtigung des Wechselkurses.

③ Die Anweisung `return` enthält die Berechnung des Preises in Euro mit der Konstanten `usd_kurs` multipliziert. Die Berechnung wird ausgeführt, bevor der Rücksprung erfolgt. Das Ergebnis dieser Berechnung wird an die Funktion `main` zurückgegeben.

④ In dieser Zeile beginnt die Funktion `main`.

⑤ Der Variablen `ergebnis` wird ein Wert zugewiesen. Dieser Wert ergibt sich aus dem Rückgabewert des Funktionsaufrufs `eur_usd`.

⑥ Sie können auch innerhalb der Funktion `printf` die Funktion `eur_usd` aufrufen und den Wert des Ergebnisses auf dem Bildschirm ausgeben.

Mit der Verwendung von Prototypen eröffnen sich die Möglichkeiten der modularen Programmierung. Bei der örtlichen Trennung von Funktionsdeklaration und Funktionsdefinition (Implementierung), z. B. in verschiedenen Dateien, können Sie den Funktionsprototypen in eine Header-Datei **.h* auslagern. Diese wird dann über den Präcompiler in die **.c*-Datei eingefügt. Sie erhalten damit eine Schnittstelle für die Funktionen über die entsprechenden Prototypen. Der Aufruf der Funktionen ist möglich, da die Namen, die Parameterlisten und die Rückgabewerte der Funktionen bekannt sind. In der dazugehörigen **.c*-Datei können Sie die Funktionen implementieren. Diese **.c*-Datei kann auch als kompilierte Objektdatei (binär) oder als Bibliothek vorliegen und somit Fremden den Einblick in die Implementierung verwehren.

5.7 Übungen

Referenzen als Parameter verwenden

Übungsdatei: --	**Ergebnisdatei: *Tausch.c***

① Schreiben Sie eine Funktion, die die Werte zweier Variablen vertauschen kann. Verändern Sie die Werte durch die Übergabe von Referenzen.

Funktionen einsetzen

Übungsdatei: --	**Ergebnisdatei: *Lohn.c***

① Schreiben Sie ein Programm, das auszugsweise eine Lohnrechnung simuliert. Dabei werden in einzelnen Funktionen nach der Übergabe des Bruttogehaltes die jeweils errechneten Kosten für Lohnsteuer, Rentenversicherung, Pflegeversicherung, Krankenversicherung etc. auf dem Bildschirm ausgegeben.

② Geben Sie außerdem den Netto- und Bruttoverdienst aus.

Referenzübergabe und globale Variablen

Übungsdatei: --	**Ergebnisdatei: *Referenz-Antwort.txt***

① Erklären Sie, wie die Verwendung globaler Variablen mithilfe von Referenzen umgangen werden kann.