

Programmierung(1)



Agenda

- Datenspeicherung auf dem Rechner: Eine vereinfachte Darstellung
- Maschinensprache versus "Höhere Sprache": Binärcode versus Quellcode
- Programmier-Paradigma: Imperative, Deklarative oder Objekt-orientierte Sprache
- Entwicklungsumgebung Code::Blocks (installieren!?): Editor, Compiler, Debugger, Linker, Ausgabe-Konsole

Der erste eigene Code

- Aufbau des Quellcodes
 - Hauptprogramm: main()
 - Anweisungsblock
 - Anweisung
 - Präprozessoranweisung: include<...>
- Variable: **Deklarieren, Definieren, Initialisieren**
- Zuweisung: Literale und/oder Ergebnisse von Rechenoperationen
- Ausgabe: printf()
 - Formatierung:
 - String-Literale
 - Typen
 - Layout



Datenspeicherung auf dem Rechner

- Die kleinste Speichereinheit auf einem Computer wird als **Bit** bezeichnet. Es handelt sich dabei technisch betrachtet um eine Art Schalter, der nur die beiden Zustände: ...
 - "geschlossen"(Strom fließt) oder
 - "offen"(Strom fließt nicht)

... annehmen kann.

Diese beiden Zustände werden der Kürze wegen gerne mit den Ziffern 1 (für geschlossen) und 0 (für offen) dargestellt.

- Solche **Bitfolgen** können ganz unterschiedlich interpretiert werden. Sie können je nach Bedarf (bzw. nach gewünschter Festlegung) als Zeichen, Zahl oder auch Befehl verstanden werden. Letzteres werden wir uns nun anschauen, wenn es im Folgenden um Befehle (einer Maschinensprache) geht, die von einem Prozessor ausgeführt werden sollen:



Maschinensprache versus "Höhere Sprache"

- Eine **Maschinensprache** (bzgl. eines bestimmten Prozessortyps) bezeichnet den gesamten Umfang all jener Befehle, die der entsprechende Prozessor unmittelbar ausführen kann.
- Maschinensprachen-Befehle werden in der Regel als Bitfolge dargestellt, deren Länge sich nach dem Prozessortyp richtet (für gewöhnlich handelt es sich bei der Länge um ein Vielfaches von 8).
- Maschinensprache-Befehle führen allerdings üblicherweise nur sehr elementare Prozesse aus. Entsprechend arbeitet ein in Maschinensprache geschriebenes Programm vergleichsweise **klein-schrittig**. Daher ist die Erstellung eines solchen Programms oft recht aufwendig.
- "Höhere" Sprachen umgehen dieses Problem, indem sie den Programmierer zunächst einen sogenannten Quellcode erzeugen lassen, der zum einen wesentlich größere Arbeitsschritte erlaubt, und zum anderen deutlich leichter lesbar ist. Da dieser Quellcode aber für einen Prozessor nicht ausführbar ist, wird er vor der Programm-Ausführung (mittels Compiler) oder während der Ausführung (mittels Interpreter) in Maschinensprache übersetzt.

Hinweis:

■ Die hohe Arbeitsgeschwindigkeit heutiger Computer macht das Codieren in Maschinensprache oft überflüssig. Immer dort aber, wo extreme Performance und/oder geringer Speicherplatzbedarf von großer Bedeutung ist, macht es Sinn, Programme (oder Teile von Programmen) in Maschinensprache zu schreiben.



Programmier-Paradigma (bzw. Philosophie, Konzept oder "Art und Weise")

- Die älteste (und vielleicht auch noch verbreitetste) Art des Programmierens ist die sogenannte imperative (alternative Bezeichnung: prozedurale) Programmierung. Sie arbeitet wie der Name schon sagt mit einer Abfolge von Befehlen und erinnert an ein Kochrezept, bei dem man der Reihe nach (bzw. "von oben nach unten") alle Handlungsschritte auflistet. Dies wird jenes Verfahren sein, mit dem wir uns (an Hand der Programmiersprache ANSI C) in diesem Baustein befassen werden.
- Eine andere Philosophie des Programmierens verfolgen die sogenannten **deklarativen** Sprachen. Diese verlangen nicht, dass man Schritt für Schritt beschreibt, "wie" ein Programm zum gewünschten Ziel gelangt. Statt dessen genügt es, dieses Ziel zu beschreiben, da der Compiler/Interpreter anschließend eigenständig entscheidet, welche Arbeitsschritte zum Erreichen des Ziels nötig seien werden. (Beispiel für eine deklarative Sprache in dieser Umschulung ist die Datenbanksprache **SQL**)
- Ein deutlich jüngeres (aber zunehmend an Bedeutung gewinnendes) Programmier-Konzept ist das der OOP (**Objekt-Orientierte-Programmierung**). Es arbeitet mit einem "Ordnungs-Prinzip", dessen Grundlagen (siehe: "Strukturen") wir auch bei ANSI C kennenlernen werden. (Beispiel für eine OOP-Sprache ist **C#**)
- Neben vieler weiterer Paradigmen, seien exemplarisch nur noch die "funktionale Programmierung" (basiert auf Rekursion) und "logische Programmierung" (insbesondere für mathematische Beweise) genannt.



Entwicklungsumgebung (IDE = Integrated Development Environment)

- Wer einen Quellcode schreibt, der könnte dies in einem ganz gewöhnlichen Editor tun. Anschließend müsste er mit einem Compiler (gegebenenfalls unter Verwendung eines Präprozessor) diesen Code übersetzen lassen, mit einem Linker die einzelne Module seines Programms zusammenführen, eventuell mittels Debugger nach Fehlern suchen und die aus all diesen Vorarbeiten resultierende Datei schließlich noch abspeichern, um sie anschließend starten bzw. ausführen zu können.
- Da dies lästig ist, wurden Entwicklungsumgebungen entwickelt, mit denen all diese Funktionalitäten in einem einzigen Programm vereinigt wurden. In der Regel reicht dann ein einziger "Klick" um die obigen Arbeitsschritte der Reihe nach abarbeiten zu lassen.
- Wir wollen in diesem Baustein mit der IDE Code::Blocks arbeiten, die Sie bitte herunterladen, sofern sich diese nicht bereits auf ihrem Rechner befindet.
 - Hinweis: Nach der Installation wird möglicherweise der Compiler nicht gefunden. In diesem Fall navigieren Sie bitte bis: Settings/Compiler/Toolchain executables klicken dort auf den Button Auto-detect und bestätigen dies anschließend mit "OK"



Vorbemerkungen:

- Der Quellcode besteht aus den im Folgenden dargestellten Abschnitten, die wir uns der Einfachheit halber "graphisch animiert" anschauen werden.
- Selbstverständlich werden wir aber nach dieser PowerPoint-Präsentation auch eine gemeinsame praktische Übung absolvieren, bevor Sie anschließend durch weitere Einzelübungen Gelegenheit bekommen, das hier Vorgestellte ausführlich zu trainieren.
- Darüber hinaus gilt, dass die am heutigen Tag vermittelten Inhalte bei praktisch allen kommenden Codierungs-Aufgaben von Bedeutung seien werden. Damit ist dann aber auch deren regelmäßige Wiederholung gesichert.



Wir beginnen mit dem eigentlichen **Hauptprogramm**, das in ANSI C stets mit dem Funktionsnamen **main()** versehen wird:

```
#include<...>
main()
         Anweisung1;
         Anweisung2;
```



Anfang und Ende des Anweisungsblocks werden (wie schon beim Pseudocode üblich) durch **geschwungene Klammern** dargestellt

```
#include<...>
main()
         Anweisung1;
         Anweisung2;
```



Hier **beginnt** der Anweisungsblock der main-Funktion:

```
#include<...>
#define ...
main()
         Anweisung1;
         Anweisung2;
```



```
#include<...>
#define ...
main()
         Anweisung1;
         Anweisung2;
```



Alle **Anweisungen** (Befehle) werden **eingerückt** notiert:

```
#include<...>
main()
           Anweisung1;
           Anweisung2;
Breite der Einrückung:
  1 Tabulator-Schritt
```



Jede Anweisung wird mit einem **Semikolon** abgeschlossen:

```
#include<...>
#define ...
main()
         Anweisung1;
         Anweisung2;
```



Um komplizierte Abschnitte eines Codes für den Leser erläutern zu können, gibt es die Möglichkeit, Kommentare in einen Quellcode einzutragen, die vom Compiler beim Übersetzen ignoriert werden.

Falls ein Code nur eine einzige Zeile benötigt, so reicht die Notation zweier **Slash-Symbole** unmittelbar vor dem Kommentar:

```
#include<...>
main()
          Anweisung1;
          Anweisung2;
         // einzeiliger Kommentar
```



Mehrzeilige Kommentare starten mit /* und enden mit */

```
#include<...>
main()
          Anweisung1;
          Anweisung2;
            mehrzeiliger
            Kommentar
```



Oberhalb der main-Funktionen werden die **Präprozessor-Anweisungen** notiert. (Sie beginnen stets mit dem **Hash-Symbol**)

```
#include<...>
#define ...
main()
         Anweisung1;
         Anweisung2;
```



Mit **include** werden benötigte Bibliotheken eingebunden (dazu heute noch mehr)

```
#include<NameDerBibliothek>
#define ...
main()
         Anweisung1;
          Anweisung2;
```



Die Präprozessor-Anweisung **define** werden wir erst später behandeln

```
#include<...>
#define ...
main()
         Anweisung1;
         Anweisung2;
```



Der Bereich unterhalb der Präprozessor-Anweisungen, aber oberhalb vom main kann für (eigene) **Funktionen** verwendet werden.

Da wir dieses Thema aber erst deutlich später behandeln, werden wir diesen Bereich zunächst ungenutzt lassen.

```
#include<...>
#define ...
  •••
main()
         Anweisung1;
         Anweisung2;
```



Aus genau diesem Grund wollen wir auch die main-Funktion zunächst so "sparsam" wie möglich notieren. Gemeint ist ...

```
#include<...>
#define ...
main()
         Anweisung1;
         Anweisung2;
```



... dass wir die hier farblich markierte Schreibweise zunächst bewusst unterlassen werden, da wir weder **Parameter**, noch den **Typ des Rückgabewertes**, geschweige denn den **Return-Befehl** bereits behandeln wollen.

```
#include<...>
#define ...
int main(int argc, char *argv)
          Anweisung1;
          Anweisung2;
          return 0;
```



Der Compiler reagiert dann zwar mit dem folgenden Hinweis:

```
warning: return type defaults to 'int' [-Wimplicit-int]
```

... dies ist aber nur eine **Warnung** (also keine Fehlermeldung) und wird von uns bis auf weiteres ignoriert. Erst wenn wir das Thema der *Funktionen* behandelt haben werden, wird es Sinn machen (zumindest Teile) der angesprochenen Notation zu ergänzen.

```
#include<...>
#define ...
main()
         Anweisung1;
         Anweisung2;
```



Der erste eigene Code – Variablen

```
main()
         int zahl=5;
         char zeichen='w';
         float x=2.355;
```



Der erste eigene Code – Variablen - Deklarieren

```
main()
         int zahl=5;
          char zeichen='w';
         float x=2.355;
```



Der erste eigene Code – Variablen - Definieren

```
main()
          int zahl=5;
          char zeichen='w';
          float x=2.355;
```



Der erste eigene Code – Variablen - Initialisieren

```
main()
         int zahl=5;
          char zeichen='w';
         float x=2.355;
```



Der erste eigene Code – Variablen – Definieren+Deklarieren

```
main()
                                 int zahl=5;
                                 char zeichen='w';
                                 float x=2.355;
Alle Variablen müssen in einem Quellcode vor der
ersten Verwendung deklariert und definiert werden!
```



Der erste eigene Code – Variablen – Definieren+Deklarieren

```
main()
                                   int zahl=5;
                                   char zeichen='w';
                                   float x=2.355;
Eine sofortige Initialisierung ist aber nicht zwingend gefordert:
                                   float kommazahl;
```



```
main()
         int a=1;
                     Erste Zuweisungen werden
         int b=2;
                     als Initialisierung bezeichnet
         int c=3;
```



```
main()
          int a=1;
          int b=2;
          int c=3;
           a=1;
                   Erst beim Überschreiben wird
           b=2;
                   von einer Zuweisung gesprochen
          c=3;
```



```
main()
                                                 int a=1;
Die Definition (und Deklaration) jeder einzelnen
                                                int b=2;
Variable geschieht natürlich stets nur einmalig
                                                 int c=3;
                                                 a=1;
                                                 b=2;
                                                 c=3;
```

```
main()
                           int a=1;
                           int b=2;
                           int c=3;
                           a=1;
                            b=2;
Keine erneute Definition!
                           c=3;
```

```
main()
                                          int a=1;
                                          int b=2;
                                          int c=3;
                                          a=1;
                                          b=2;
                                          c=3;
                                          a = a + 1;
Und natürlich können auch die Resultate
                                          b = 2*b
von Rechnungen zugewiesen werden:
```

Der erste eigene Code – Ausgabe – String

Code-Ausschnitt printf("Hallo Welt"); Mit der Funktion printf können Texte auf der Konsole ausgeben werden. Solche Texte werden auch als "Zeichenkette" oder "String" bezeichnet.

Konsolen-Ausgabe

Hallo Welt



Der erste eigene Code – Ausgabe – String

Code-Ausschnitt printf("Hallo Welt"); Um dem Compiler deutlich zu machen, wo ein String anfängt und wo er aufhört, muss er stets in (doppelten) Anführungszeichen notiert werden!

Konsolen-Ausgabe

Hallo Welt



Der erste eigene Code – Bibliothek – stdio.h

Code-Ausschnitt

printf("Hallo Welt");

Die Funktion **printf** ist (zumindest der reinen Lehre nach) dem Compiler unbekannt.

Der Compiler weiß also nicht, was er zu tun hat, wenn er den Befehl printf kompilieren soll. Daher muss zunächst eine **Bibliothek** eingeführt werden, in der dies "erklärt" wird.

Es gibt eine große Anzahl von Bibliotheken, die die unterschiedlichsten Funktionen erklären. Die Bibliothek, in der printf erklärt wird, heißt **stdio.h** (Abkürzung für STanDard Input Output)

Konsolen-Ausgabe

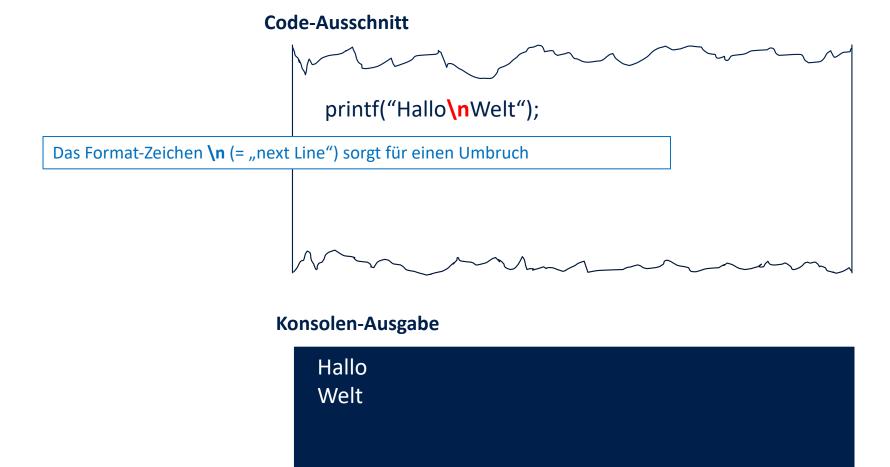
Hallo Welt

Quellcode

```
#include<stdio.h>
main()
          printf(...)
```

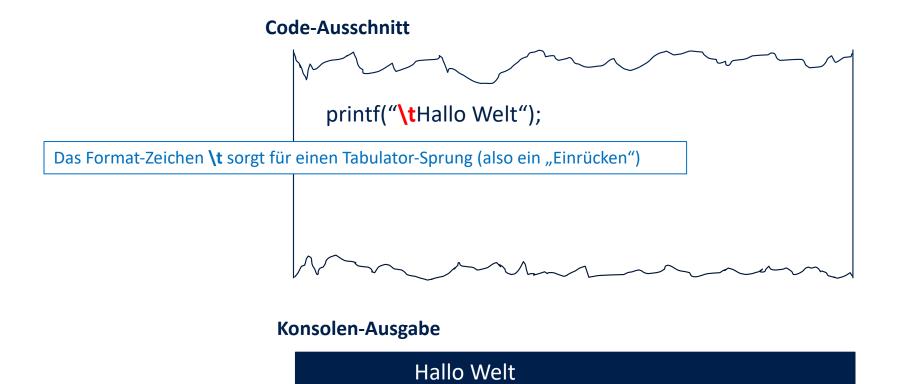


Der erste eigene Code – Ausgabe-Formate: Zeilenumbruch



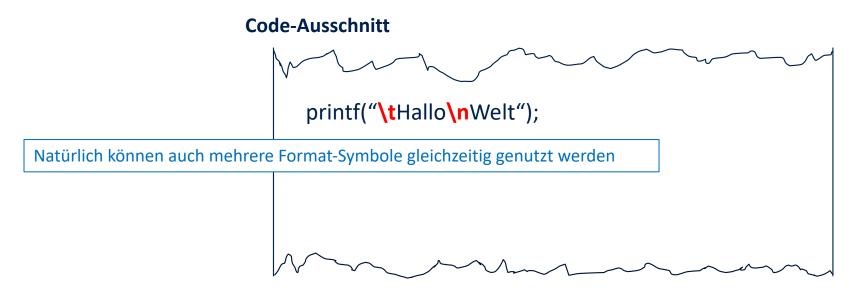


Der erste eigene Code – Ausgabe-Formate: Tabulator





Der erste eigene Code – Ausgabe-Formate: mehrere



Konsolen-Ausgabe

Hallo Welt



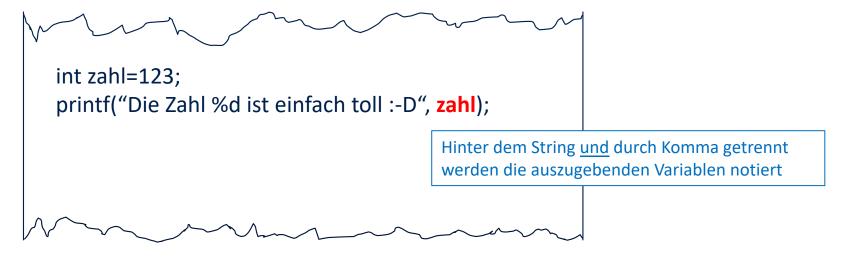


Konsolen-Ausgabe

Die Zahl 123 ist einfach toll :-D



Code-Ausschnitt



Konsolen-Ausgabe

Die Zahl 123 ist einfach toll :-D



Code-Ausschnitt

```
int i=100;
char c='g';
float f=5234.57;
printf("%d%c Gold kosten %f Euro", i, c, f);

Natürlich können auch mehrere Variablen
(beliebigen Typs) ausgegeben werden
```

Konsolen-Ausgabe

100g Gold kosten 5234.570000 Euro



Code-Ausschnitt

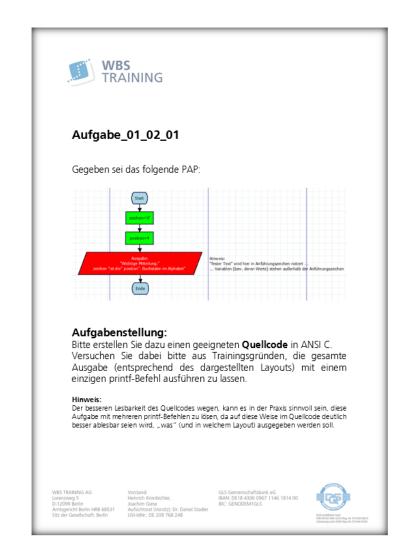
```
int i=100;
char c='g';
float f=5234.57;
printf("%d%c Gold kosten %.2f Euro", i, c, f);
```

Konsolen-Ausgabe

100g Gold kosten 5234.57 Euro



Der erste eigene Code – Gemeinsame Übung A_01_02_01







VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!









