Eine praktische Einführung

in die Programmiersprache C

Alle Programm-Codes und Inhalte

sind im Internet abrufbar:

www.c-howto.de

Autor

Elias Fischer

Diplom-Informatiker (FH)

Kontakt

post@c-howto.de

Druckversion

v1.2 (Oktober 2008)

Inhaltsverzeichnis

1 Einführung

Hier beginnt das C Tutorial. In diesem Kapitel werden grundlegende Begriffe zur Programmierung erläutert. Weiters wird das erste Programm in C geschrieben und gezeigt, wie das Programmieren unter Linux und Windows funktioniert.

1.1 Programmieren

Bevor man mit dem Programmieren beginnt, sollte man natürlich auch wissen, wozu man Dinge "programmiert". In diesem Lehrbuch befassen wir uns mit dem Programmieren von Computern. Ein Computer ist zuerst nur ein Haufen Silizium mit gelöteten Chips. In dieser Form ist ein Computer völlig funktionslos - er kann nichts. Ein Computer muss erst programmiert werden, bevor er einen Nutzen darstellt. Der Mensch programmiert den Computer derart, dass dieser ähnlich wie ein Hund, auf Kommandos reagiert. Wir halten also fest: Wir müssen etwas zuerst programmieren, damit etwas das macht, was wir wollen.

1.2 Programm?

Programmiersprache

Wenn wir einen Hund trainieren, verwenden wir die die Sprache des Menschen (wie z.B. "Sitz", "Beifuß"), da der Mensch nicht bellen möchte. Wenn wir einen Computer programmieren, verwenden wir eine Programmiersprache, da wir nicht in Nullen und Einsen sprechen wollen. Eine Programmiersprache ist also für Computer und Mensch verständlich. Der Mensch muss eine Programmiersprache wie eine Fremdsprache lernen. Beherrscht er sie, kann er damit ein Programm schreiben. Damit der Computer dieses ausführe kann, übersetzt der Computer das Programm in seine "Muttersprache", welche aus Nullen und Einsen besteht, erst dann kann es gestartet und benutzt werden.

Programm

Ein Programm ist ein Ablauf von Aktionen, um ein Ziel zu erreichen. Wollen wir ein Brot backen, so müssen wir Menschen folgendes dafür tun:

1. Alle Zutaten einkaufen (Eier, Mehl, Hefe, ...)
2. Die Zutaten zu einem Teig verarbeiten
3. Den Teig in eine Form geben und einige Zeit in den Ofen geben

Dies war ein Programm für Menschen. Ein Computer-Programm besteht aus Befehlen, welche der Computer ausführen soll. Müssten wir zum Beispiel ein Programm für einen Roboter schreiben, der unser Brot backen soll, könnten die Befehle folgendermaßen heißen:

1. Prüfe ob alle Zutaten vorhanden sind
2. Wenn eine Zutat fehlt, geh diese einkaufen
3. Zerschlage das Ei in einer Schüssel
4. Gebe Mehl in die Schüsssel
5. Gebe Hefe in die Schüsssel
6. Gebe Wasser in die Schüsssel
7. Verrühre die Zutaten in der Schüssel zu einem Teig
8. Gebe den Teig in eine Form
9. Stelle den Ofen an
10. Warte bis der Ofen 230 Grad heiß ist
11. Stelle die Form in den Ofen

Wie wir sehen ist das Programm für den Computer wesentlich länger und detaillierter, als das Programm für den Menschen. Das liegt daran, dass der Mensch eine gewisse Intelligenz und Erfahrung besitzt, ein Computer hingegen hat davon gar nichts. Wir Menschen müssen uns um alles kümmern, d.h. wir müssen jeden Schritt ganz genau angeben und alle Fehler berücksichtigen, welche passieren könnten. Für jeden Fehler muss man wiederum eine Ausweichaktion parat haben. Eine Ausweichaktion für den Fehler "das Brot zu lange im Ofen gelassen" wäre zum Beispiel, dass wir ein Brot in der Bäckerei kaufen würden.

1.3 Warum C?

Unsere Programmiersprache, mit der wir das Programmieren lernen möchten, ist C. C ist einfach aufgebaut und kann sehr viel. Deshalb ist die Sprache auch weit verbreitet und wird in den meisten Bereichen eingesetzt. Wenn man C kann, hat man es wesentlich einfacher weitere Programmiersprachen wie C++, Java, Perl oder PHP zu erlernen, da die Sprachen gewisse Ähnlichkeiten aufweisen.

1.4 Hello World

Endlich ist es soweit - wir schreiben unser erstes Programm. Es kann zwar nicht viel, aber es gibt immerhin einen Text auf dem Bildschirm aus, nämlich "Hello World". Im hellgrauen Kasten sehen wir immer den Quellcode, also unser Quellprogramm, welches wir in der Programmiersprachen C erstellt haben:

```
#include<stdio.h>

int main() {
    printf("Hello World\n");
    return 0;
}
```

Wenn wir unserem Computer sagen, es solle dieses Programm ausführen, sehen wir folgendes auf dem Bildschirm. Die sogenannte Bildschirmausgabe wird hier in einem dunkelgrauen Kasten dargestellt.

```
Hello World
```

Das hier jetzt "Hello World" auf dem Bildschirm steht, haben wir dem Befehl **printf** zu verdanken. In dem Befehlsnamen steckt das Wort "print", was übersetzt "drucken" heißt. Dass der Befehl nun etwas auf den Bildschirm "druckt", bzw. ausgibt, ist also nicht sehr verwunderlich.

1.5 Der Anfang

1.5.1 Texteditor

Wenn wir unsere C Programme schreiben, genügt ein einfacher Text-Editor. Von Vorteil wäre ein Syntax Highlighting Feature, welches bestimmte Textblöcke farbig darstellt. Unter Linux eignet sich Kate sehr gut dafür. Unter Windows kann man Notepad++ oder Scite verwenden.

1.5.2 Grundgerüst

Wenn man sich für einen Texteditor entschieden hat, erstellt man eine neue Datei mit der Endung ".c" und gibt das Grundgerüst ein:

```
#include<stdio.h>

int main(void) {
    return 0;
}
```

Im ersten Teil des Gerüstes binden wir die Bibliotheken ein, hier z.B. **stdio.h**. Eine Bibliothek benötigen wir, um Befehle in unserem Programm verwenden zu können. In einer Bibliothek sind Befehle zu einem bestimmten Thema dokumentiert. In der **stdio.h** sind die Standard-Befehle zur Ein- und Ausgabe, dazu aber später mehr. Danach sehen wir die Zeile **int main()**, wobei **main** unser Hauptprogramm

kennzeichnet. Unser eigentliches Programm schreiben wir zwischen die geschweiften Klammern **{ }**. Im Grundgerüst haben wir momentan nur einen Befehl stehen: **return 0;**, dieser wird auch später erklärt. Wichtig hierbei ist, dass wir alle Befehle mit einem **;** (Strichpunkt) abschließen. Die Wörter **include**, **int** und **return** sind sogenannte Schlüsselwörter, das heißt sie gehören zum Wortschatz der Programmiersprache. Immer wenn wir sie verwenden, nehmen wir eine spezielle Funktion der Programmiersprache in Anspruch. Im Kapitel 1.6 Compiler wird erklärt, wie man das Programm ausführt.

1.6 Compiler

Wie wir bereits gelernt haben, gibt es Programmiersprachen deshalb, weil wir die Sprache des Computers (Nullen und Einsen) nicht effektiv beherrschen können. Das heißt, wir schreiben unser Programm in einer Programmiersprache, auch Hochsprache genannt. Diese wird jedoch nicht vom Computer verstanden und muss also in Nullen und Einsen übersetzt werden. Dies ist Aufgabe des **Compilers**:

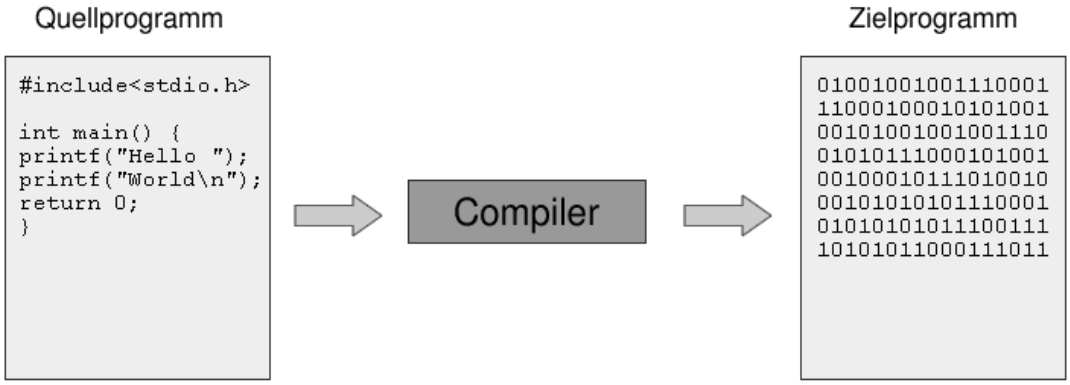


Illustration 1: Arbeitsweise eines Compilers

Der Inhalt des Zielprogramms besteht nur aus zwei unterschiedlichen Zeichen ("0" und "1"), deshalb nennt man es auch Binärprogramm (binär, lat. "aus zwei Einheiten bestehend"). Im Englischen würde man es "binaries" oder "excuteable" (ausführbar) nennen. Das Quellprogramm nennt man auch Quelltext, kurze Programme "Skript"; im Englischen "Source" oder "Source Files" ("Source" = Quelle, "File" = Datei), bzw. "Script".

1.6.1 Aufgaben des Compilers

Beim Übersetzungsvorgang, auch "Kompilierung" genannt, hat der Compiler folgende Aufgaben. Die Fehler die dabei auftreten können, wurden anhand des Programmbeispiels Hello World Programm veranschaulicht.

- **Lexikalische Analyse**: Hier prüft der Compiler ob er auch alle Wörter im Quellprogramm versteht, bzw., ob wir Schreibfehler gemacht haben. Würden wir z.B. nur "print" statt "printf" schreiben, ist das ein Fehler.
- **Syntaktische Analyse**: Hier wird die Grammatik unseres Quellprogramms geprüft. Würden wir statt "printf("hello");" nur "printf "hello";" schreiben, ist das ein Fehler.
- **Semantische Analyse**: Hier wird geprüft, ob unser Quellprogramm überhaupt einen Sinn ergibt. Unser printf Befehl z.B., benötigt einen Text, den er ausgeben kann. Geben wir ihm aber nur eine Zahl, wie "printf(4711);", wäre das falsch.

Findet der Compiler einen Fehler, wird die Kompilierung nicht abgeschlossen. Der Compiler gibt die Fehlermeldungen mit Zeilennummern aus, so hat man es nicht ganz so schwer die Fehler in seinem Programm zu finden.

1.6.2 Linux

Wenn wir ein fertiges Quellprogramm haben, ist es ein leichtes dieses unter Linux zu kompilieren. Man benötigt ein Terminalfenster (Shell) und wechselt in das Verzeichnis, in dem das C Script liegt. Nehmen wir an, es heißt "hello.c", geben wir ein:

```
# gcc hello.c
```

Danach sollte bei erfolgreicher Komiplierung eine neue Datei in dem gleichen Verzeichnis liegen, namens "**a.out**". Das ist nun unsere Binärdatei, welche wir ausführen können. Dies geht mit:

```
# ./a.out
```

Hätte man jetzt unser Hello World Programm kompiliert und ausgeführt, würde der Text "Hello World" am Bildschirm erscheinen. Beim Kompilieren kann man auch direkt einen Namen für die Ausgabedatei festlegen, sodass diese nicht immer "a.out" heißt. Hier würde die Ausgabedatei "hello" heißen:

```
# gcc hello.c -o hello
```

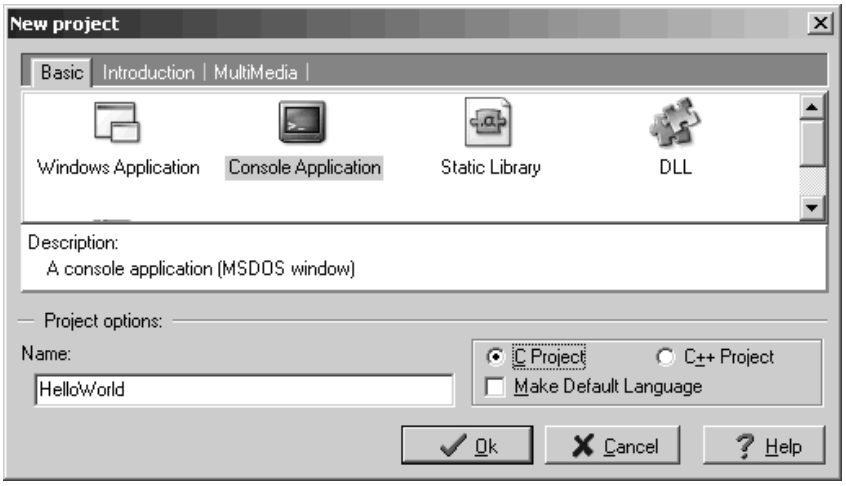
1.6.3 Windows

Für das Kompilieren unter Windows wird hier das Programm **Dev-C++** vorgestellt. Es bietet eine grafische Benutzeroberfläche und ist kostenlos verfügbar.

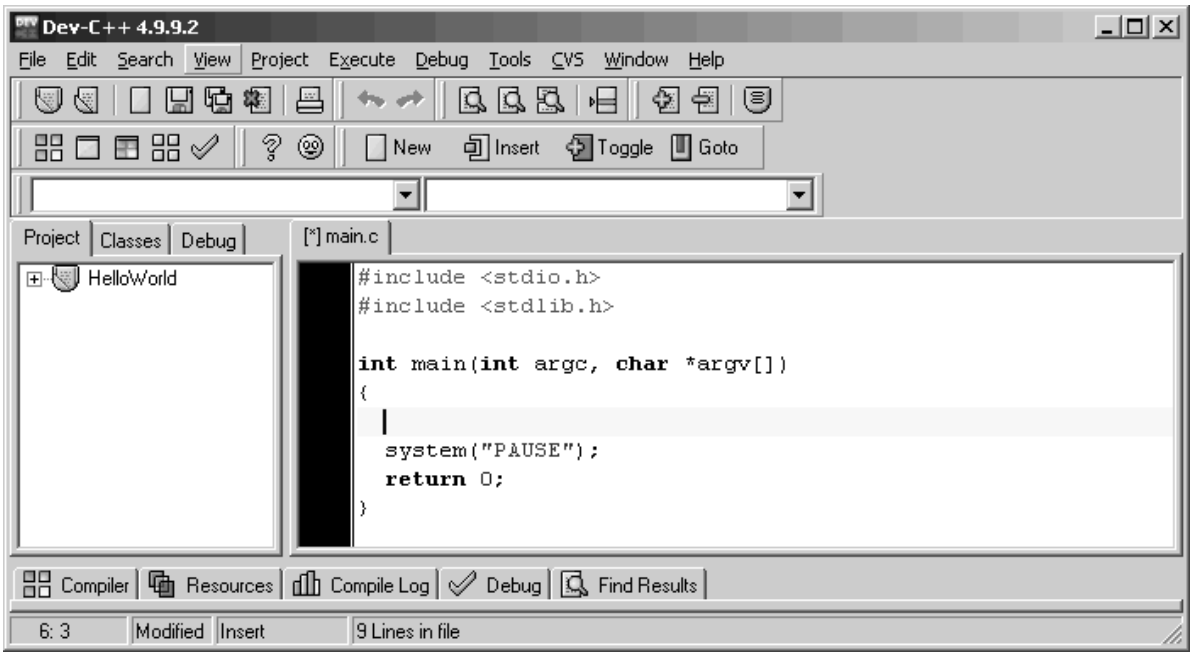
Dev-C++ kann unter http://www.bloodshed.net/devcpp.html mit einem Klick auf "Goto Download Page" heruntergeladen werden. Danach wird die Datei einfach ausgeführt und die Installation mit den vorgegebenen Standard-Werten durchgeführt. Programmiert wird meist in englischer Sprache, deshalb wurde der Compiler auch in Englisch installiert. Damit deine Ansicht mit der Anleitung übereinstimmt, sollte gleich am Anfang die Sprache **Englisch** ausgewählt werden.

Nach der Installation kann man **Dev-C++** starten. Wir wollen nun das Kompilieren mit **Dev-C++** mit einem HelloWorld-Programm testen. Wir klicken dazu auf **File -> New -> Project** um ein neues Projekt zu erstellen und erhalten die folgende Ansicht, welche wir mit diesen Werten ausfüllen:

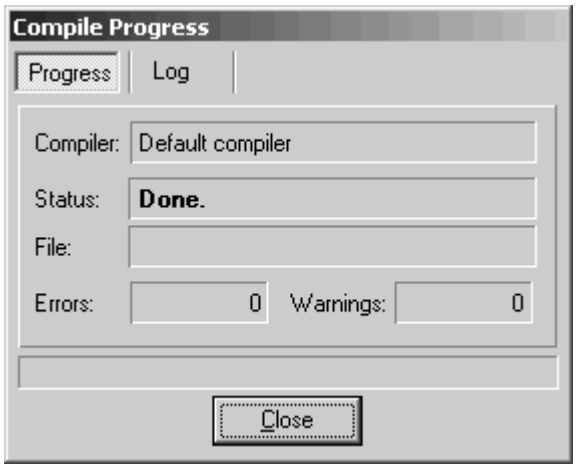
- **Console Application**, erstellt eine einfache Konsolenanwendung für die Eingabeaufforderung
- **C-Project**, erstellt ein C-Programm
- **Name**, der Name des Programms, z.B. HelloWorld



Nach einem Klick auf **OK** geben wir noch einen Speicherort für das neue Projekt an. Danach erhalten wir bereits ein lauffähiges Programm. Im nächsten Bild ist bereits die Zeile markiert, in der wir unseren eigenen Quelltext schreiben können. Die Zeile **system("PAUSE");** am Ende des Quelltextes bewirkt, dass sich das Programm am Ende nicht sofort schließt, sondern noch auf einen Tastendruck wartet und somit die Programm-Ausgabe sichtbar bleibt.

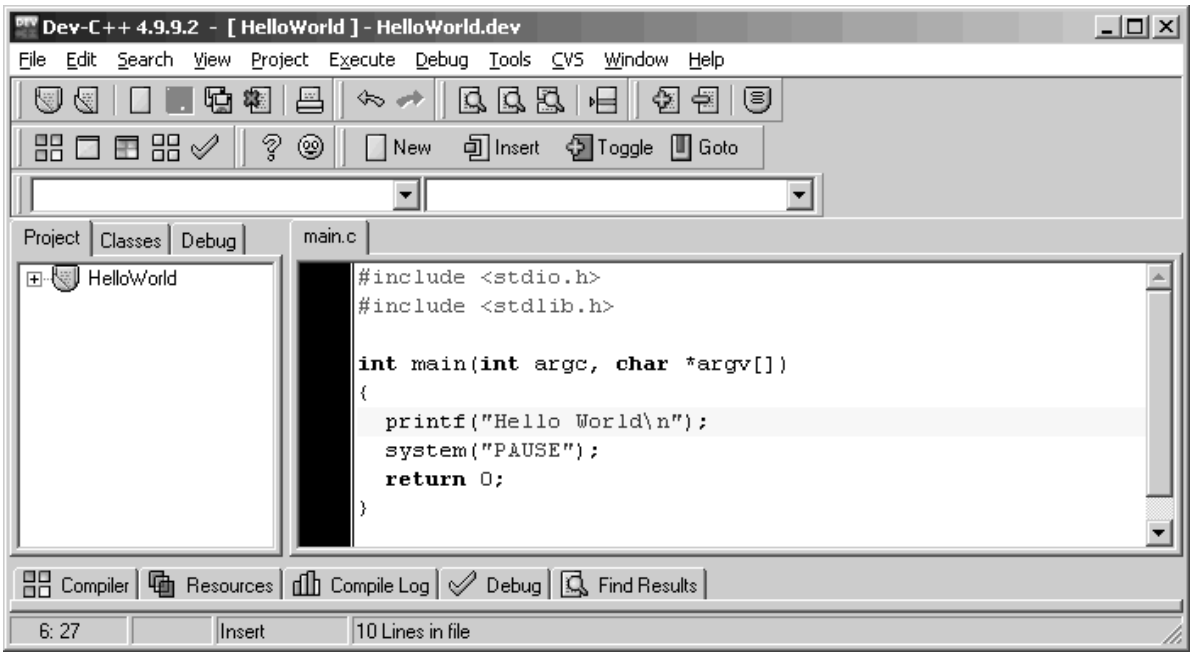


Wir wollen nun dieses Programm kompilieren, dies geht über **Execute -> Compile**. Danach erscheint ein Fenster mit Status **Done**.



Nachdem das Programm kompiliert ist, können wir es über **Execute -> Run** ausführen. Wir werden noch nach dem Speicherort der Quelltext-Datei gefragt, diese können wir in dasselbe Verzeichnis wie die Projektdatei plazieren. Der Standard-Dateiname ist **main.c**. Die Programm-Ausgabe erscheint in einem Terminal-Fenster mit schwarzem Hintergrund. Aufgrund der Programmzeile **system("PAUSE");** bleibt es solange geöffnet, bis eine beliebige Taste gedrückt wird.

Wir können nun beginnen die Programm-Vorlage zu individualisieren und eine Ausgabe mit "Hello World" einfügen. Alternativ kann man auch den Quelltext von unserem HelloWorld-Programm einfügen.



Die Schritte Kompilieren und Ausführen lassen sich über **Execute -> Compile & Run** auch mit einem Klick ausführen. Noch schneller geht dies mit der Taste **F9**.

1.6.4 C-HowTo Code-Beispiele

Die Beispiele in diesem HowTo wurden für Linux verfasst, sie sind aber auch mit einer Ausnahme unter Windows lauffähig. Damit man wie schon erwähnt die Bildschirmausgabe sieht, sollte beim Kompilieren mit Dev-C++ immer **zusätzlich** die Zeile **system("PAUSE");** am Programmende vor die **return** Anweisung gesetzt werden.

```
    ...
    system("PAUSE");
    return 0;
}
```

1.7 Kommentare

Eine wichtige und grundlegende Sache bei der Programmierung sind die Kommentare. Wird ein Programm komplexer verliert man schnell die Übersicht. Schaut man sein Programm nach ein paar Monaten nochmals an, weiß man auf Anhieb gar nicht mehr, was es eigentlich macht. Kommentare sollen hierbei helfen komplizierte Programmabschnitte kurz und prägnant zu erklären. Kommentare werden vom Compiler komplett ignoriert, da sie nur für den Menschen eine Funktion haben. In den Kommentaren gelten also keinerlei Schreibrichtlinien.

1.7.1 Einzeilige Kommentare

Ein einzeiliges Kommentar wird duch einen Doppel-Slash gemacht.

```
#include<stdio.h>

int main() {
    // Hier wird ein Text ausgegeben:
    printf("Hello World\n");
    return 0;
}
```

1.7.2 Mehrzeilige Kommentare

Mehrzeilige Kommentare sind sinnvoll, wenn man eine längere Beschreibung in den Quelltext packen möchte. Weiters kann man größere Blöcke im Quelltext schnell auskommentieren, sodass diese vom Compiler nicht mehr verarbeitet werden, was bei einer Fehlersuche hilft. Das Kommentar wird mit **/*** eingeleitet und endet mit ***/**. Alles was dazwischen ist, wird vom Compiler ignoriert.

```
/*
    Hier beginnt
    ein langes Kommentar
    und hier
    endet es
*/

#include<stdio.h>

int main() {
    /*
    printf("Hello ")
    printf("World\n");
    */

    // Hier wird ein Text ausgegeben:
    printf("Servus\n");

    return 0;
}
```

Hier wird nun nicht wie erwartet "Hello Word" ausgegeben, sondern "Servus".

1.8 Binärsystem

Der Umgang mit dem binären Zahlensystem gehört zum Allgemeinwissen eines Informatikers und trägt zum Verständnis der Computer-Materie bei. Das Binärsystem wird auch Dualsystem oder Zweiersystem genannt. Der Name kommt daher, dass dieses Zahlensystem nur zwei verschiedene Ziffern für die Darstellung von Zahlen verwendet: **0** und **1**. **Null** ist **0** und **Eins** ist **1**, aber was ist **Zwei**? Da wir nun keine Ziffern mehr zur Verfügung haben, müssen wir durch die Position der 0- und 1-Ziffern weitere Zahlen darstellen. Hier sind die Zahlen von 1 bis 10 im dezimalen Zahlensystem und den zugehörigen Binärwerten dargestellt. Binärzahlen können auch mit führenen Nullen aufgefüllt werden, dies ändert den Wert nicht (0001 = 1).

```
dez    binär

0         0
1         1
2        10
3        11
4       100
5       101
6       110
7       111
8      1000
9      1001
10     1010
```

Der Binärwert **10** steht also für die **Zwei**. Zur Umrechnung in einen dezimalen Wert kann untere Skala hilfreich sein. Wir sehen, dass die Skala von rechts nach links mit Zweier-Potenzen von 2 hoch 0 bis 2 hoch 7 steigt. Darunter sehen wir den jeweiligen Dezimalwert. Nun legen wir die den Binärwert **10** an das rechte Ende, was der Anfang der Skala ist, sodass die 0 bei 2^0 und die 1 bei der 2^1 ist. Jetzt müssen nur noch alle Werte addiert werden, an denen eine **1** steht, was hier nur **2^1** ist, also 2. Für die Zahl Zehn ist der Binärwert **1010**, sodass eine **1** an **2^3** und **2^1** steht, 8 + 2 ist 10.

```
2^7    2^6    2^5    2^4    2^3    2^2    2^1    2^0

128    64    32      16     8      4      2      1
```

1.8.1 Umrechnung Dezimal nach Binär

Für die umgekehrte Umrechnung kann ebenfalls die Skala verwendet werden - oder das Horner-Schema. Hierbei teilen wir die Dezimalzahl und notieren den Rest, dann wird das Ergebnis wieder geteilt und der Rest notiert. Dies geschieht solange die zu teilende Zahl Null ist. Danach ergibt der Rest von unten nach oben gelesen die Binärzahl.

```
Bsp. 1: 6 dezimal zu binär

6 : 2 = 3 Rest 0
3 : 2 = 1 Rest 1
1 : 2 = 0 Rest 1

Ergebnis: 110


Bsp. 2: 23 dezimal zu binär

23 : 2 = 11 Rest 1
11 : 2 =  5 Rest 1
5 : 2 =  2 Rest 1
2 : 2 =  1 Rest 0
1 : 2 =  0 Rest 1

Ergebnis: 10111
```

1.9 Hexadezimalsystem

Das Binärsystem hat die Basis 2 und besteht somit aus zwei verschiedenen Ziffern. Das Hexadezimalsystem hat die Basis 16 und besteht somit aus 16 verschiedenen Ziffern. Mit den Dezimalzahlen 0 bis 9 können wir bereits 10 Ziffern darstellen, für die restlichen 6 Ziffern werden die ersten Buchstaben des Alphabets verwendet.

```
dez   hex

0     0
1     1
2     2
3     3
4     4
5     5
6     6
7     7
8     8
9     9
10    A
11    B
12    C
13    D
14    E
15    F
```

Für die Berechnung von Zahlen zwischen den Zahlensystemen wird wieder die Skala verwendet, wobei wir beim Hexadezimalsystem die Basis 16 verwenden. Möchte man **E7** hex nach Dezimal umrechnen, geschieht dies über 14 * 16^1 + 7 * 16^0 = 14 * 16 + 7 * 1 = 224 + 7 = 231. In der oberen Tabelle ist zu sehen, dass E für den dezimalen Wert 14 steht.

```
16^4    16^3    16^2    16^1    16^0

65536   4096    256     16      1
```

Für die Umrechnung von Dezimal nach Hex kann wieder das Horner-Schema verwendet werden, nur dass dann nicht durch 2 sondern durch 16 geteilt wird. Um hexadezimale Zahlen erkenntlich zu machen sind folgende Schreibweisen üblich: E7 hex, E7h, E7H und 0xE7. In der C-Programmierung verwenden wir die Schreibweie beginnend mit "Null x", also z.B. 0xE7.

1.10 Bits und Bytes

Ein Bit (binary digit) ist die kleinste darstellbare Maßeinheit für Informationen. Als **binary digit** steht es für **eine** Speicherstelle für einer Binärzahl. Die Binärzahl **1010** hat demnach 4 Bits. Da ein Bit nur die zwei Zustände (0 und 1) darstellen kann, ist durch **n** Bits die Darstellung von **2^n** verschiedenen Binärzahlen möglich. Bei 2 Bits ergibt das 2^2 = 4 verschiedene Zahlen, bzw. Zustände: 00, 01, 10 und 11.

1.10.1 Bytes, Kilobytes und Megabytes

Dies sind größere Maßeinheiten, welche auf Bits basieren. Im Sprachgebrauch und in den meisten Fällen ist mit **1 Byte** gleich **8 Bits** gemeint. Die Übertragungsgeschwindigkeiten von Internet-Providern werden zum Beispiel immer in Bit angegeben. Demnach entspricht eine Geschwindigkeit von 256 kBit/s also 32 kByte/s, der Wert wird einfach durch 8 geteilt. Bei der Umrechnung zwischen Byte, Kilobyte & Co wird in der Computertechnik der Faktor 1024 verwendet. In den Klammern ist auch die gebräuchliche und standardisierte Abkürzung für die Maßeinheit angegeben - die Groß- und Kleinschreibung ist relevant.

```
1 Byte              =  8 Bit
1 Kilobyte (kB)     =  1024 Bytes
1 Megabyte (MB)     =  1024 Kilobytes
1 Gigabyte (GB)     =  1024 Megabytes
1 Terabyte (TB)     =  1024 Gigabytes
1 Petabyte (PT)     =  1024 Terabytes
1 Exabyte (EB)      =  1024 Petabytes
1 Zettabyte (ZB)    =  1024 Exabytes
1 Yottabyte (YB)    =  1024 Zettabytes
```

Nach dem Internationalen Einheitensystem (SI) ist der Umrechnungsfaktor jedoch 1000. Dies machen sich Festplattenhersteller zunütze, welche diesen Faktor bei der Angabe der Festplattenkapazität verwenden. Demnach sind 250 GB nicht 250 * 1024 = 256.000 MB, sondern 250.000 MB. Man muss also mit weniger rechnen, als angegeben ist.

2 Anhang

2.1 ASCII Tabelle

Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zeichen	Scan-code	ASCII hex	dez	Zeichen
	00	0	NUL		20	32	SP		40	64	@	0D	60	96	`
	01	1	SOH ^A	02	21	33	!	1E	41	65	A	1E	61	97	a
	02	2	STX ^B	03	22	34	"	30	42	66	B	30	62	98	b
	03	3	ETX ^C	29	23	35	#	2E	43	67	C	2E	63	99	c
	04	4	EOT ^D	05	24	36	$	20	44	68	D	20	64	100	d
	05	5	ENQ ^E	06	25	37	%	12	45	69	E	12	65	101	e
	06	6	ACK ^F	07	26	38	&	21	46	70	F	21	66	102	f
	07	7	BEL ^G	0D	27	39	'	22	47	71	G	22	67	103	g
0E	08	8	BS ^H	09	28	40	(	23	48	72	H	23	68	104	h
0F	09	9	TAB ^I	0A	29	41	)	17	49	73	I	17	69	105	i
	0A	10	LF ^J	1B	2A	42	*	24	4A	74	J	24	6A	106	j
	0B	11	VT ^K	1B	2B	43	+	25	4B	75	K	25	6B	107	k
	0C	12	FF ^L	33	2C	44	,	26	4C	76	L	26	6C	108	l
1C	0D	13	CR ^M	35	2D	45	-	32	4D	77	M	32	6D	109	m
	0E	14	SO ^N	34	2E	46	.	31	4E	78	N	31	6E	110	n
	0F	15	SI ^O	08	2F	47	/	18	4F	79	O	18	6F	111	o
	10	16	DLE ^P	0B	30	48	0	19	50	80	P	19	70	112	p
	11	17	DC1 ^Q	02	31	49	1	10	51	81	Q	10	71	113	q
	12	18	DC2 ^R	03	32	50	2	13	52	82	R	13	72	114	r
	13	19	DC3 ^S	04	33	51	3	1F	53	83	S	1F	73	115	s
	14	20	DC4 ^T	05	34	52	4	14	54	84	T	14	74	116	t
	15	21	NAK ^U	06	35	53	5	16	55	85	U	16	75	117	u
	16	22	SYN ^V	07	36	54	6	2F	56	86	V	2F	76	118	v
	17	23	ETB ^W	08	37	55	7	11	57	87	W	11	77	119	w
	18	24	CAN ^X	09	38	56	8	2D	58	88	X	2D	78	120	x
	19	25	EM ^Y	0A	39	57	9	2C	59	89	Y	2C	79	121	y
	1A	26	SUB ^Z	34	3A	58	:	15	5A	90	Z	15	7A	122	z
01	1B	27	Esc	33	3B	59	;		5B	91	[		7B	123	{
	1C	28	FS	2B	3C	60	<		5C	92	\		7C	124	|
	1D	29	GS	0B	3D	61	=		5D	93	]		7D	125	}
	1E	30	RS	2B	3E	62	>	29	5E	94	^		7E	126	~
	1F	31	US	0C	3F	63	?	35	5F	95	_	53	7F	127	DEL

2.2 Schlüsselwörter

Die Schlüsselwörter der Programmiersprache C.

- asm
- auto
- break
- case
- char
- const
- continue
- default
- do
- double
- else
- enum
- extern
- float
- for
- goto
- if
- int
- long
- register
- return
- short
- signed
- sizeof
- static
- struct
- switch
- typedef
- union
- unsigned
- void
- volatile
- while

3 Abbildungsverzeichnis