Die Programmiersprache C

Allgemeines

C ist eine kompakte, strukturierte, typisierte, hardware-nahe, portable, höhere Programmiersprache;
die Anzahl eingebauter Befehle und reservierter Schlüsselwörter und Symbole ist klein;
die Ein-/Ausgabe ist nicht im Befehlsumfang enthalten, sondern wird in Bibliotheken bereitgestellt.

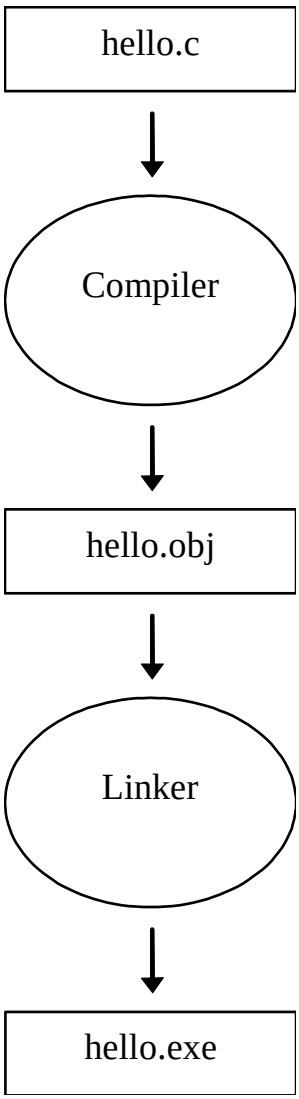
Compiler – Linker

Ein kleines C-Programm

Jedes C-Programm besteht zumindest aus einer Funktion *main*, die immer den Rückgabewert int haben sollte. Im folgenden Beispiel gibt das Programm den Text „Hello world.“ aus.

```
#include <stdio.h>
int main (void) {
  printf("Hello world!");
  return 0;
}
```

Um zu einem ausführbaren Programm zu kommen, muss der Quellcode (Sourcecode), geschrieben und gespeichert in einer Datei mit der Endung .c, übersetzt werden. Dabei entsteht eine sogenannte „Object-Datei“. Diese Datei enthält die Übersetzung in Maschinencode. Das „Object-File“ wird gelinkt, sprich Informationen aus inkludierten Bibliotheken wird hinzugefügt, wodurch ein „Executable“ entsteht, d.h. eine ausführbare Datei.



Grundsätzlich gilt

- C-Quellcode muss zur Grammatik (Syntax) von C konform sein, der Quellcode muss die fachliche Anforderung realisieren (Semantik).
- Ein C-Programm startet mit der Funktion main.
- C unterscheidet wie Java zwischen Groß- und Kleinschreibung! Wird Main statt main geschrieben, so liegt ein Syntaxfehler vor.
- C-Quellcode setzt sich zusammen aus Quellcode-Dateien (Endung: .c), die Header-Dateien einbinden (Endung: .h);
- Mit #include werden fremde Programmteile hinzugefügt. Dies entspricht der Angabe import aus Java.
- Das Semikolon (‘;’) beendet wie in Java eine Anweisung. Dennoch gibt es kleine Unterschiede. Hinter #include steht zum Beispiel kein Semikolon!
- Quellcode muss vom Compiler übersetzt und dessen Ausgabe vom Linker mit Bibliotheken zu ausführbarem Programm gebunden werden;
- Der Compiler ruft vor der eigentlichen Übersetzung den Präprozessor auf, der die Präprozessoranweisungen (beginnen mit #) auswertet.

Kommentare

Quellcode, der nicht übersetzt werden soll, kann als Kommentar markiert werden (// markiert bis Zeilenende, /* bis erstem folgendem */)

Reservierte Schlüsselwörter

Folgende Namen sind in C reserviert und dürfen nicht für die Benennung von Variablen, Konstanten, Typen bzw. Funktionen verwendet werden:

Variablendeklaration:

char, short, int, long, float, double, void; signed, unsigned; volatile, auto, register, static, const, extern

Typdeklaration:

struct, union, enum, typedef

Ablaufsteuerung:

if, else, switch, case, default, break, do, while, for, continue, goto, return

Typ-/Variablengröße bestimmen:

sizeof()

Literale – unmittelbare Werte

Im Programm unmittelbar zu verarbeitende Werte (Zahlen, Buchstaben, Texte) nennt man Literale.

Zeichen

Ein einzelnes Zeichen schließt man in einfache Hochkommata ein, zum Beispiel:

```
'A'
```

Das Zeichen ' kann nur als `'\''` geschrieben werden, d.h. dem Zeichen ' muss mit dem vorangestellten Backslash die Hochkomma-Bedeutung genommen werden.

Zeichenketten / Strings

Zeichenketten sind in C als Felder von Zeichen realisiert und müssen in doppeltes Hochkomma eingeschlossen werden. Eine Zeichenkette entspricht dabei einer Folge von Buchstaben, die nach dem letzten Zeichen das Zeichen '\0' als Endmarke enthält.

```
"Hallo!"
```

Im Speicher wird dies abgelegt als Byte-Folge:

´H´	´a´	´l´	´l´	´o´	´!´	´\0´

Sonderkonstanten

Für Steuerzeichen in Strings sind sogenannte Sonderkonstanten definiert:

Konstante	**ASCII-Wert**	**Bedeutung**
\n	10	Zeilenvorschub
\t	9	Tabulatorsprung (horizontal)
\v	11	Tabulatorsprung (vertikal)
\b	8	Backspace
\r	13	Zeilenende
\f	12	neue Seite (form feed)
\0	0	String-Ende
\\	92	\-Zeichen
\'	39	´-Zeichen
\"	34	“-Zeichen

Beispiele:

Zeichenkette	entspricht
`"\n"`	(Leerzeile)
`"Hallo,\nwie geht's."`	Hallo, wie geht's.
`"Vorname\tNachname"`	Vorname Nachname

Konstanten

Im Gegensatz zu Variablen, können sich konstante Werte während ihrer gesamten Lebensdauer nicht ändern. Dies kann etwa dann sinnvoll sein, wenn Konstanten am Anfang des Programms definiert werden, um sie dann nur an einer Stelle im Quellcode anpassen zu müssen.
Ein Beispiel hierfür ist etwa die Mehrwertsteuer. Wird sie erhöht oder gesenkt, so muss sie nur an einer Stelle des Programms geändert werden. Um einen bewussten oder unbewussten Fehler des Programmierers zu vermeiden, verhindert der Compiler, dass der Konstante ein neuer Wert zugewiesen werden kann.
In der ursprünglichen Sprachdefinition von Dennis Ritchie und Brian Kernighan (K&R) gab es nur die Möglichkeit, mit Hilfe des Präprozessors symbolische Konstanten zu definieren. Dazu dient die Präprozessoranweisung #define. Sie hat die folgende Syntax:

```
#define IDENTIFIER token-sequence
```

Präprozessoranweisungen werden nicht mit einem Semikolon abgeschlossen!

```
#define MWST 20
```

Bewirkt, dass jede vorkommende Zeichenkette MWST durch die Zahl 20 ersetzt wird. Eine Ausnahme besteht lediglich bei Zeichenketten, die durch Anführungszeichen eingeschlossen sind:

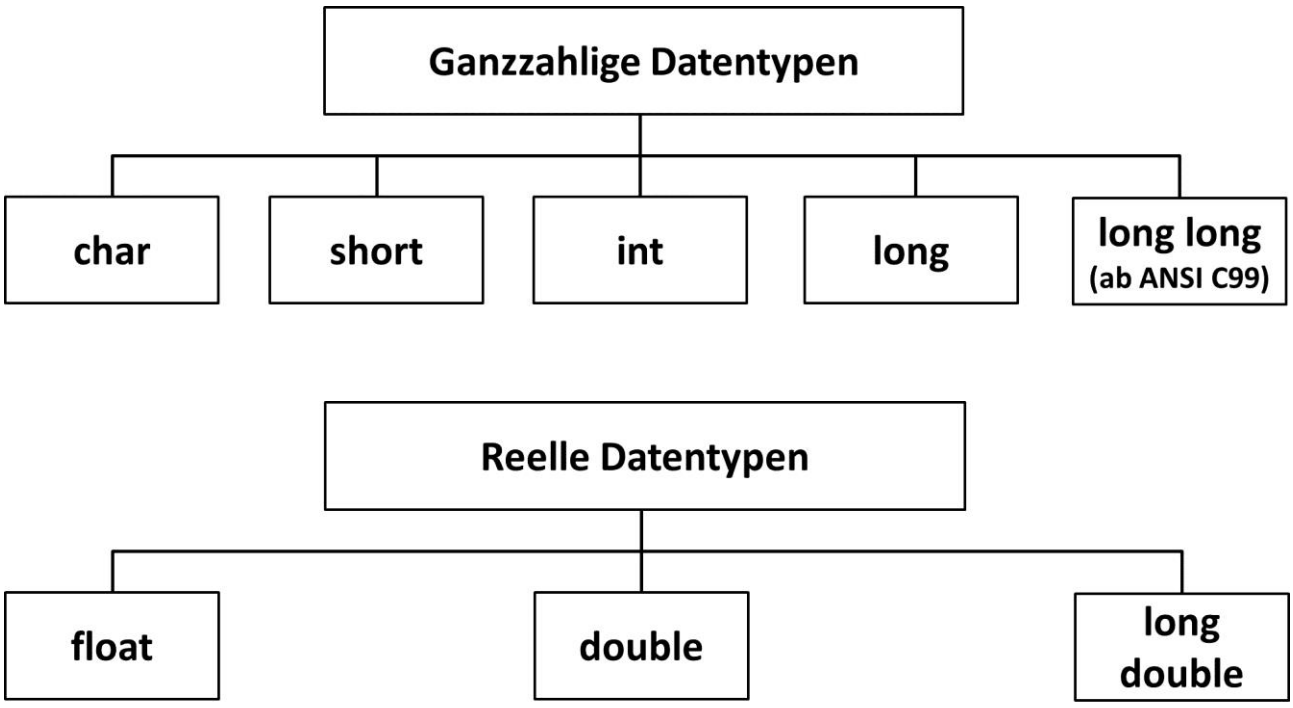
```
"Die aktuelle MWST"
```

Hierbei wird die Zeichenkette MWST nicht ersetzt.
Die Großschreibung ist nicht vom Standard vorgeschrieben. Es ist kein Fehler, anstelle von MWST die Konstante MwSt oder mwst zu benennen. Allerdings benutzen die meisten Programmierer Großbuchstaben für symbolische Konstanten.
Bei einer Definition durch #define wird Wort für Wort ohne Beachtung der Syntax und der Semantik übersetzt.
Die Definition durch const hingegen folgt einer Syntax, die mit der Definition einer Variablen identisch ist. Es muss der Typ der Konstante (Standard int) angegeben werden. Die Konstante muss, wenn sie deklariert ist, unbedingt initialisiert werden, da es unmöglich ist, den Wert einer Konstanten außerhalb des Initialisierungsvorgangs zu verändern.

```
const int MWST=20;
```

Zahlen

Zahlen werden im Quellcode unmittelbar geschrieben und vom Compiler als entsprechendes Bitmuster in den Maschinencode des erzeugten Programms eingetragen. Bei negativen Zahlen muss ein – vorangestellt werden.
Gleitkommazahlen werden am Dezimalpunkt oder an der Exponentenangabe (e bzw. E) erkannt und können in normaler Darstellung oder in Exponentialschreibweise notiert werden. Dabei kann das e klein oder groß geschrieben werden. Neben dezimalen Zahlen können auch oktale bzw. hexadezimale Zahlen verwendet werden, wobei oktale Zahlen mit einer Null und hexadezimale Zahlen mit einer Null gefolgt von einem großen oder kleinen x beginnen müssen.
Dezimale Zahlen haben den Typ int bzw. long, falls int zu klein ist. Oktale bzw. hexadezimale Zahlen bekommen den kleinsten Typ aus der Reihe int, unsigned int, long und unsigned long.
Durch Anhängen eines oder mehrerer Buchstaben kann der Typ der Zahl erzwungen werden. L bzw. l erzwingen long, UL bzw. ul erzwingen unsigned long, U bzw. u erzwingen unsigned int und f erzwingt float.
Beispiele:
12, -2345, 45.349, 34609UL, 0278, 0377, 0x378, 0xaa55, 0xffff, -78.19304e+12, 1.5035e-3



Ein-/Ausgabe (E/A)

Es gibt keine C-Befehle für die Ein-/Ausgabe. Die Ein-/Ausgabe wird in C über Bibliotheksfunktionen zur Verfügung gestellt. Die wichtigsten E-/A-Funktionen sind printf und scanf.

Ausgeben mit printf

Die Funktion printf dient zur Ausgabe von Werten. Um die Funktion verwenden zu können, muss am Anfang des Programmes *#include <stdio.h>* stehen. Dieser Funktion können eine beliebige Anzahl von Parametern übergeben werden. Der erste Parameter muss ein Formatstring sein. Dahinter folgen die auszugebenden Werte.

Syntax

```
printf ( Formatstring, ... )
```

Zur Ausgabe von Variableninhalten dienen in C und C++ Formatangaben, die in der Formatzeichenkette zusammen mit dem auszugebenden Text stehen. Diese Formatangaben beginnen immer mit dem Prozentzeichen (‘%’). Diesem Zeichen folgen detaillierte Angaben zur Ausgabeform der Variable. Die Variablennamen selbst stehen als Argumente, durch Komma getrennt, hinter dem Formatstring.

Die Syntax der Formatangabe lautet:

```
% [ Flag ] [ Breite ] [ . Präzision ] [ Präfix] Typ
```

Die in eckigen Klammern stehenden Ausdrücke sind wahlfrei. Sie haben im einzelnen folgende Bedeutung (keine vollständige Aufzählung):

Flag:	-	linksbündige Ausgabe (standardmäßig: rechtsbündig)
	+	Pluszeichen vor positiven Zahlen wird ausgegeben
	0	Führende Zeichen werden mit Nullen aufgefüllt
	#	Vor Oktal- und Hexadezimalzahlen wird ‘0’ bzw. ‘0x’ geschrieben. Gleitkommazahlen haben immer einen Dezimalpunkt.
Breite:	Zahl	Die angegebene Zahl gibt die Mindestbreite der Ausgabe an.
	*	Die nächste Zahl der Argumentliste wird als Breite interpretiert (int-Zahl).
Präzision:	Zahl	Anzahl der Nachkommastellen bei Gleitkommazahlen, Anzahl der maximal auszugebenden Zeichen bei Zeichenketten.
	*	Die nächste Zahl der Argumentliste wird als Präzision interpretiert (int).
Präfix:	l	Variable wird als Typ (unsigned) long interpretiert und ausgegeben.
	L	Variable wird als Typ long double interpretiert und ausgegeben.
	h	Variable wird als Typ short interpretiert und ausgegeben.

Typ:	d, i	Dezimalzahl (dezimal, integer)
	o	Dezimalzahl oktal ausgeben
	x, X	Dezimalzahl hex ausgeben mit Klein-/Großschreibung der Buchstaben
	u	Dezimalzahl ohne Vorzeichen (unsigned)
	c	einzelnes Zeichen (char)
	f	Gleitpunktzahl im double-Format (float)
	e, E	Gleitpunktzahl in Exponentialdarstellung (Klein-/Großschreibung)
	g, G	Gleitpunktzahl, bei kleinem Exponenten f-Darstellung, sonst e bzw. E
	s	Zeichenkette (String)
	p	Zeigerausgabe (Pointer)
	%	**Ausgabe des Prozentzeichens**

Beispiel

```
#include <stdio.h>

int main (void) {

  char c;
  float f;
  int i;

  c = 'A';
  f = 839.21;
  i = 40;

  printf ("Ausgabe eines Integers:\n");
  printf ("i = |%d|\n",i);
  printf ("i = |%5d|\n",i);
  printf ("i = |%-5d|\n",i);
  printf ("i = |%5.3d|\n",i);
  printf ("i = |%1d|\n\n",i);

  printf ("Ausgabe eines Characters:\n");
  printf ("c = |%c|\n",c);
  printf ("c = |%5c|\n",c);
  printf ("c = |%-5c|\n",c);
```

```
  printf ("c = |%d|\n",c);
  printf ("c = |%0c|\n\n",c);

  printf ("Ausgabe eines Floats:\n");
  printf ("f = |%f|\n",f);
  printf ("f = |%10.3f|\n",f);
  printf ("f = |%10.3e|\n",f);
  printf ("f = |%-10g|\n",f);
  printf ("f = |%2.1f|\n\n",f);
  return 0;

}
```

Erzeugte Ausgabe:

```
Ausgabe eines Integers:
i = |40|
i = |   40|
i = |40   |
i = |  040|
i = |40|

Ausgabe eines Characters:
c = |A|
c = |    A|
c = |A    |
c = |65|
c = |A|

Ausgabe eines Floats:
f = |839.210022|
f = |   839.210|
f = | 8.392e+02|
f = |839.21    |
f = |839.2|
```

Einlesen mit scanf

Dient zum Einlesen von Variablenwerten. Auch hier muss ein Format-String angegeben werden. Die Reihenfolge und Datentypen der Variablen müssen den Formatelementen exakt entsprechen.
Achtung: mit der Eingabe kann keine gleichzeitige Ausgabe verbunden werden.

Beispiele:

```
/* scanf wartet auf Zahl */
scanf ("%d", &i);
/* scanf wartet auf zwei ints und einen char */
scanf ("%d %d %c", &i, &j, &k);
```

Achtung:

Scanf erwartet die Adresse der Variablen, um die Werte hineinstellen zu können. Adress-Operator & nicht vergessen!
Die Zuordnung von Zeichen zu einer Variable endet beim ersten Zwischenraumzeichen (Leerzeichen, Tabulator bzw. Zeilenende) bzw. beim ersten Zeichen, das nicht zur Typangabe passt (Buchstabe bei Zahlen, jedes Zeichen außer Ziffern bei Ganzzahlen,...). Das erste nicht zugeordnete Zeichen bleibt im Eingabepuffer für die weitere Eingabeverarbeitung. Das Entfernen von ungelesenen Zeichen kann durch fflush(stdin) erzwungen werden. %c liest als einziges Formatelement auch Zwischenraumzeichen, d.h. %s liest nur ein Wort bis zum ersten Zwischenraumzeichen. Bei scanf ist keine Angabe der Genauigkeit möglich. Eine Angabe der Feldbreite begrenzt die Anzahl der für eine Variable betrachteten Zeichen (siehe Beispiel für Datumseingabe).

Bsp.:

Einlesen von Datum und Uhrzeit im Format tt-mm-jjjj/hh:mm:ss (z.B. 30-11-2010/14:24:04).

```
scanf("%2u-%2u-%4u/%2u:%2u:%2u", &tag, &monat,
&jahr, &stunde, &minute, &sekunde);
```

Zeichenkette einlesen mit fgets

Mit fgets(zielzeichenkette, zielzeichenkettenlaenge, stdin) kann eine Zeichenkette aus der Standardeingabe befüllt werden, wobei bis zum abschließenden Zeilenendezeichen '\n' gelesen wird. Die Zeichen vor dem Zeilenendezeichen werden in die Zielzeichenkette übertragen und mit einem String-Endezeichen '\0' abgeschlossen. Fgets übernimmt nur maximal zielzeichenkettenlaenge – 1 Zeichen, sodass ein Zeichenkettenüberlauf verhindert wird.

Funktion getchar und putchar (Modul: <stdio.h>)

Lesen eines Zeichens von Standardeingabe und gleichzeitige Ausgabe am Bildschirm:

```
int getchar(void);
```

Schreiben eines Zeichens auf die Standardeingabe:

```
putchar(int c);
```

Abfrage des Eingabeendes (Rückgabewert –1 bei getchar); wird entweder durch Datei-Ende bei Eingabeumleitung erreicht oder durch Drücken von Control-Z:

```
#define EOF -1
c= getchar ();
if (c != EOF) putchar (c);
```

Zeichenweises Kopieren von Standard-Eingabe auf Ausgabe:

```
#define EOF -1
int main (void) {
  int i=getchar();
  while (i != EOF) {
    putchar (i);       /* zeilenorientiert ! */
    i=getchar();
  }
  return 0;
}
```

ANSI-C-Datentypen und Formatelemente (Wertebereich bezieht sich auf gnu C-Compiler von mingw):

Typ	Byte	Werte-bereich von	Werte-bereich bis	Genauigkeit	Format-elemente	Beispiel

char	1	-128	127		%c	char zeichen='a';
						// zeichen mit ASCII-Code initialisieren
						printf("%c", zeichen);
						// Buchstaben ausgeben
						scanf("%c", &zeichen);
						// Buchstaben einlesen

unsigned char	1	0	255		%c	char zeichen='a';
						// zeichen mit ASCII-Code initialisieren
						printf("%c", zeichen);
						// Buchstaben ausgeben
						scanf("%c", &zeichen);
						// Buchstaben einlesen

signed char	1	-128	127		%c	char zeichen='a';
						// zeichen mit ASCII-Code initialisieren
						printf("%c", zeichen);
						// Buchstaben ausgeben
						scanf("%c", &zeichen);
						// Buchstaben einlesen

short	2	-32768	32767		%hd, %hi	short minizahl=-23;
						// minizahl mit -23 initialisieren
						printf("%hd", minizahl);
						// minizahl dezimal ausgeben
						scanf("%hd", &minizahl);
						// minizahl dezimal einlesen

unsigned short	2	0	65535		%hu, %ho, %hx, %hX	unsigned short minizahl2=12;
						// minizahl2 mit 12 initialisieren
						printf("%hu", minizahl2);
						// minizahl2 dezimal ausgeben
						scanf("%hu", &minizahl2);
						/* minizahl2 dezimal einlesen
						bei %ho erfolgt Ein-/Ausgabe oktal
						bei %hx bzw. %hX erfolgt Ein-/Ausgabe hexadezimal
						scanf liest bei %hu auch Eingaben mit vorangestelltem - */

int	4	-2147483648	2147483647		%i, %d	int zahl=-12345; // zahl mit -12345 init.
						printf("%d", zahl); // zahl dezimal ausgeben
						scanf("%d", &zahl);
						// zahl dezimal einlesen

unsigned int	4	0	4294967295		%u, %o, %x, %X	unsigned int zahl2=56789;
						// zahl2 mit 56789 initialisieren
						printf("%u", zahl2);
						// zahl2 dezimal ausgeben
						scanf("%u", &zahl2);
						// zahl2 dezimal einlesen
						/* bei %o erfolgt Ein-/Ausgabe oktal
						bei %x bzw. %X erfolgt Ein-/Ausgabe hexadezimal */

long	4	-2147483648	2147483647		%ld	long maxizahl=-456;
						// maxizahl mit -456 initialisieren
						printf("%ld", maxizahl);
						// maxizahl dezimal ausgeben
						scanf("%ld", &maxizahl);
						// maxizahl dezimal einlesen

unsigned long	4	0	4294967295		%lu, %lo, %lx, %lX	unsigned long maxizahl2=56789;
						// maxizahl2 mit 56789 initialisieren
						printf("%lu", maxizahl2);
						// maxizahl2 dezimal ausgeben
						scanf("%lu", &maxizahl2);
						// maxizahl2 dezimal einlesen
						/* bei %lo erfolgt Ein-/Ausgabe oktal
						bei %lx bzw. %lX erfolgt Ein-/Ausgabe hexadezimal */

long long	8	-9223372036854775808	9223372036854775807		%lld,	longling maxizahl3=-456;
						// maxizahl3 mit -456 initialisieren
						printf("%lld", maxizahl3);
						// maxizahl3 dezimal ausgeben
						scanf("%lld", &maxizahl3);
						// maxizahl3 dezimal einlesen

unsigned long long	8	0	18446744073709551615		%llu, %llo, %llx, %llX	unsigned long long zahl4=56789;
						// zahl4 mit 56789 initialisieren
						printf("%llu", zahl4);
						// zahl4 dezimal ausgeben
						scanf("%llu", & zahl4);
						// zahl4 dezimal einlesen
						/* bei %llo erfolgt Ein-/Ausgabe oktal
						bei %llx bzw. %llX erfolgt Ein-/Ausgabe hexadezimal */

float	4	1.17E-38	3.4E38	6 Stellen	%f, %e, %g	float gleitzahl=-12345.67;
						// gleitzahl mit -12345.67 initialisieren
						printf("%f", gleitzahl); // gleitzahl ausgeben
						scanf("%f", &gleitzahl);
						/* gleitzahl einlesen
						bei %e wird wissenschaftliche Notation -12.34567E+3,... verwendet
						bei %g wird die jeweils kürzere (%f oder %e) verwendet */

double	8	2.2E-308	1.8E308	15 Stel-len	%lf, %le, %lg	float gleitzahl2=-1.234567E-4;
						// gleitzahl mit -0.00001234567 initialisieren
						printf("%lf", gleitzahl);
						// gleitzahl ausgeben
						scanf("%lf", &gleitzahl);
						// gleitzahl einlesen
						/* bei %le wird wissenschaftliche Notation -1.234567E-4,... verwendet
						bei %lg wird die jeweils kürzere (%lf oder %le) verwendet */

char[]					%s	Char text[100] = "Hello world";
						/* Zeichenkette mit Platz für 99 nutzbare Zeichen anlegen und mit Hello world initialisieren */
						printf("%s", text);
						fgets(text, maxZeichenAnzahl);
						/* scanf("%s", text) liest nur bis 1. Trenn zeichen, fgets ersetzt abschließendes '\n' durch '\0' */

Das Formatelemente x bzw. X erzeugen kein Präfix 0x für die ausgegebene Hex-Zahl, sondern geben nur die Hex-Ziffern der Zahl aus.

Anmerkungen zur Thematik mit/ohne Vorzeichen

Beim Datentyp char (ohne unsigned/signed Angabe) legt der ANSI-Standard nicht fest, ob er mit (signed) oder ohne (unsigned) Vorzeichen realisiert werden muss. Der Wertebereich entspricht enttweder dem von unsigned char oder signed char.
short, int und long sind ohne Zusatz unsigned mit Vorzeichen, können durch den unsigned Zusatz aber ohne Vorzeichen behandelt werden.
float und double sind immer mit Vorzeichen, sie können nicht ohne Vorzeichen betrachtet werden.
Die Byteanzahl der einzelnen Datentypen bzw. Variablen kann mit dem Operator sizeof() ermittelt werden (Bsp. unsigned int laenge = sizeof(long);). Die Grenzen der Wertebereiche (INT_MAX, UINT_MIN,...) kann aus den Header-Dateien limits.h und float.h gewonnen werden.

Allgemeines zur Byteanzahl laut ANSI-Standard

char < short <= int <= long <= long long (short kann also höchstens so lang wie int oder long oder long long sein, int ist mindestens so lang wie short und höchstens so lang wie long oder long long sein); die oben angeführten Byteanzahlen gelten für die von der HTL Wels zur Verfügung gestellten Systeme (Windows, gnu-C in mingw).
Durch Verwendung eines falschen Formatelements im printf-Aufruf wird der Inhalt

einer Variable falsch interpretiert. D.h. es wird zum Beispiel im Fall eines unsigned int die höchste mögliche Zahl bei %d als -1 ausgegeben.

Variablen/Funktionen

Für Variablen und Funktionen werden in C üblicherweise Kleinbuchstaben genutzt. Die Namen können beliebig lang gewählt werden, jedoch werden in ANSI C nur die ersten 31 Zeichen unterschieden.

Variablendeklaration

Der Typ einer Variable wird in C vor dem Variablennamen geschrieben. Hinter dem Typ folgen ein oder mehrere Variablennamen, die durch Beistriche getrennt werden.

Syntax

Typ Variablenname {"," Variablenname}

Beispiele

```
int  i;
long j,k;
char c;
```