

Übersicht über einige Elemente von C

Zeichenvorrat von C

Der C-Compiler unterscheidet Groß- und Kleinschreibung:

Variable ≠ **variable** ≠ **VARIABLE** ≠ **vAriable**

Die 26 Großbuchstaben des englischen Alphabets:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- Die 26 Kleinbuchstaben des englischen Alphabets:

a b c d e f g h i j k l m n o p q r s t u v w x y z

- Die 10 Ziffern:

0 1 2 3 4 5 6 7 8 9

- 30 Sonderzeichen (einschließlich des Leerzeichens (blank)):**

^ ! " % & / { ([]] = } ? \ + * ~ # ' < > | , ; . : - _

- 4 nicht druckbare Zeichen:**

horizontaler Tabulator (*horizontal tab*), vertikaler Tabulator (*vertical tab*),
Seitenvorschub (*form feed*), Zeilenvorschub (*line feed*)

Diese und das Leerzeichen erzeugen beim Ausdruck leere Zwischenräume.
Im Englischen nennt man sie deshalb *white spaces*.

- 32 reservierte Schlüsselwörter:**

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
void	unsigned	volatile	while			

Diese Schlüsselwörter verwendet der C-Compiler für Vereinbarungen und zur Programmablaufsteuerung. Sie dürfen nicht anders, z.B. für eigene Namen verwendet werden!

Der C-Compiler unterscheidet Groß- und Kleinschreibung auch bei den Schlüsselwörtern: **auto** und **AUTO** haben unterschiedliche Bedeutung: **auto** ist ein Schlüsselwort **AUTO** ist beispielsweise eine Variable!

C-Datentypen und Speicherbedarf

Elementare Datentypen:

char	Byte (<i>character</i>), kann ein Zeichen aus dem Zeichensatz enthalten
int	ganze Zahl (<i>integer</i>), natürliches Format des verwendeten Prozessors
float	einfach genaue Gleitkommazahl (<i>floating point</i>)
double	doppelt genaue Gleitkommazahl (<i>double precision</i>)

Varianten:

short	kurz
long	lang
hyper	64 Bit lang
signed	mit Vorzeichen (2-Komplementzahl)
unsigned	ohne Vorzeichen (positive Zahl)

Ganzzahlige Variable (*integer*) können als rein positive Zahlen (vorzeichenlos = **unsigned**) oder als 2-Komplementzahlen (vorzeichenbehaftet = **signed**) definiert werden. Ist kein Vorzeichenformat angegeben, so wird **signed** angenommen. Bei den Typen **short int** und **long int** darf das Schlüsselwort **int** entfallen.

MSVC 6.0 Datentypen:

Datentyp	Wortlänge / [Bit]	Zahlbereich	Dezimalstellen
unsigned char	8	0 .. 255	$0 \dots 2^8 - 1$
signed char	8	-128 .. 127	$-2^7 \dots 2^7 - 1$
unsigned int	32	0 .. 4 294 967 295	$0 \dots 2^{32} - 1$
signed int	32	-2 147 483 648 .. 2 147 483 647	$-2^{31} \dots 2^{31} - 1$
unsigned short	16	0 .. 65 535	$0 \dots 2^{16} - 1$
signed short	16	-32 768 .. 32 767	$-2^{15} \dots 2^{15} - 1$
unsigned long	32	0 .. 4 294 967 295	$0 \dots 2^{32} - 1$
signed long	32	-2 147 483 648 .. 2 147 483 647	$-2^{31} \dots 2^{31} - 1$
unsigned hyper¹	64	0 .. 18446744073709551616	$0 \dots 2^{64} - 1$
signed hyper¹	64	-9223372036854775808 .. +9223372036854775807	$-2^{63} \dots 2^{63} - 1$
float	32	ca. $1.18 \cdot 10^{-38} \dots 3.4 \cdot 10^{38}$	6
double	64	ca. $2.23 \cdot 10^{-308} \dots 1.8 \cdot 10^{308}$	15
long double	64	ca. $2.23 \cdot 10^{-308} \dots 1.8 \cdot 10^{308}$	15

¹ Für das Format **hyper** muss in MSVC++ 6.0 **#include <rpc.h>** eingebunden sein!

Exakte Definitionen der Anzahl der signifikanten Stellen, sowie der Maximal- und Minimalwerte der Datentypen befinden sich in den C-Standarddateien **limits.h** bzw. **float.h**.

Namen in C

Namen in C dürfen nicht mit einer Ziffer beginnen. Als Anfangszeichen sind alle Buchstaben und der Unterstrich erlaubt. Als weitere Zeichen dürfen alle Buchstaben und Ziffern und der Unterstrich verwendet werden.

Lokale Namen:

Lokale Namen sind nach *ANSI C* verschieden, wenn sie sich in den ersten 31 Zeichen unterscheiden. Die Länge eines Namens, die ein Compiler auswertet (signifikante Länge) ist compilerabhängig. In *MSVCC++ 6.0* beträgt die signifikante Länge 247 *ASCII*-Zeichen.

Groß- und Kleinschreibung wird bei lokalen Namen unterschieden:

Variable $\neq$ variable $\neq$ VARIABLE $\neq$ vAriable

Globale Namen:

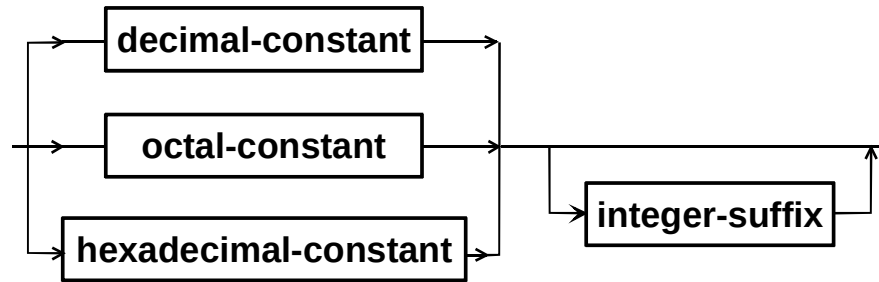
Globale Namen sind verschieden, wenn sie sich in den ersten 6 Zeichen unterscheiden.

Groß- und Kleinschreibung ist für globale Namen ohne Bedeutung:

Variable = variable = VARIABLE $\neq$ vAria1

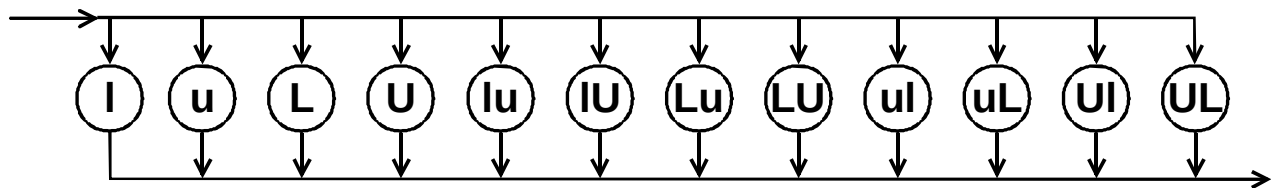
Konstanten in C

Ganzzahlige Konstanten (*integer-constant*)



Ganzzahlige Konstanten können als Dezimalzahl (*decimal-constant*) (Basis 10, Ziffern **0...9**), als Oktalzahl (*octal-constant*) (Basis 8, Ziffern. **0...7**) oder als Hexadezimalzahl (*hexadecimal-constant*) (Basis 16, Ziffern. **0..7, a...f**) angegeben werden, gefolgt von einer optionalen Nachsilbe (*integer-suffix*) für die Festlegung der Genauigkeit.

Dezimalzahlen dürfen nicht mit einer **0** beginnen, Oktalzahlen immer mit einer 0 beginnen, Hexadezimalzahlen müssen mit **0x** beginnen. Wenn nichts anderes angegeben ist, werden Ganzzahlkonstanten als **short** gespeichert., Die Nachsilbe darf jede Kombination von l und u in Groß- oder Kleinschreibung aufweisen:



Dabei bedeutet **L** oder **l**: **long**-Format und **U** oder **u**: **unsigned**-Format (vorzeichenlos).

Beispiele für ganzzahlige Konstanten

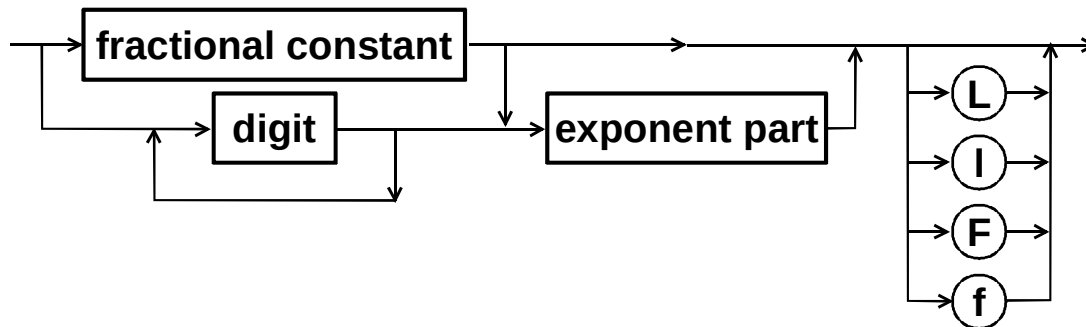
Dezimalkonstanten: **1, -987, 987u, 1234567UL, -352l**

Oktalkonstanten: **0, 0345, 0774u, 077777LU**

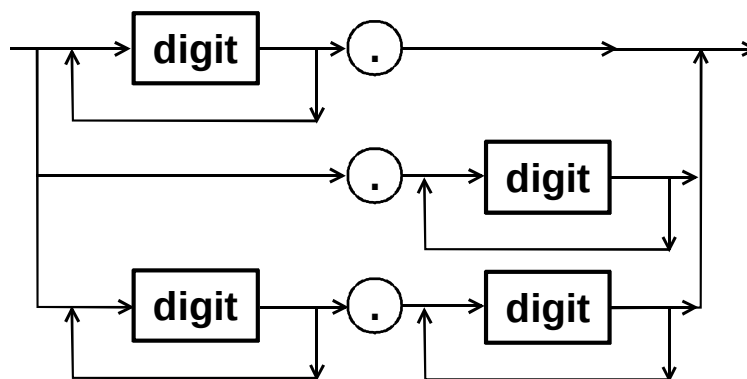
Hexadezimalkonstanten: **0x1357, 0x0AFFEu, 0xcaffeeL**

Gleitkomma-Konstanten (*floating-constant*)

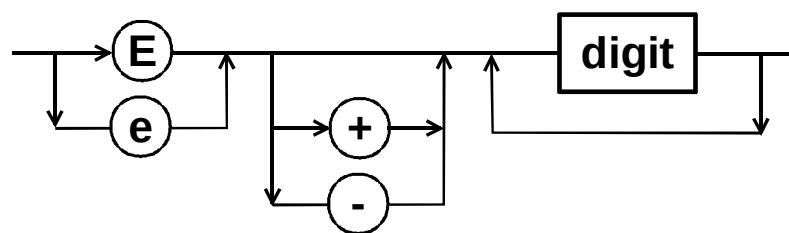
Gleitkommakonstanten beinhalten immer den Dezimalpunkt. Sie bestehen aus der Mantisse (*fractional constant*), dem Exponenten (*exponent part*, optional) und einer optionalen Nachsilbe (*floating suffix*) zur Festlegung der Genauigkeit:



In der Mantisse muss zusammen mit dem Dezimalpunkt mindestens eine Ziffer auftauchen. Der Mantisse kann eines der Vorzeichen + oder - vorangestellt werden. Ist kein Vorzeichen angegeben, so wird ein positives Vorzeichen angenommen:



Der Exponent wird immer mit ohne Leerzeichen mit **e** oder **E** eingegeben. Das Vorzeichen ist optional. Ist kein Vorzeichen angegeben, so wird ein positives Vorzeichen angenommen. Es folgen die Ziffern der 10er-Potenz. Dies entspricht der wissenschaftlichen Schreibweise, z.B. $1.25 \cdot 10^2$:



Gleitkommakonstanten werden, wenn nichts anderes angegeben ist, als **double** gespeichert

Beispiele für Gleitkommakonstanten:

1.25, **.1**, **324.**, **.987**, **1.234567L**, **352.E12I**, **1.234567L**, **.234567e-22f**

Zeichen-Konstanten (*character-constant*)

Zeichenkonstanten belegen immer nur 1 Byte. Sie können entweder als Zahl oder als *ASCII*-Zeichen eingegeben werden und sind in Apostrophe (' ') eingeschlossen. Sie dürfen jedes *ASCII*-Zeichen außer Backslash (\), Apostroph (') oder Zeilenende (*ASCII*: LF) enthalten.

Escape-Sequenzen dienen der Eingabe der folgenden 11 Sonderzeichen:

- \a** *Alert* (Piepton)
- \b** *Backspace* (←-Taste)
- \f** *Form feed* (Zeilenvorschub)
- \n** *New line* (neue Zeile)
- \r** *Return* (Wagenrücklauf)
- \t** *Horizontal Tab* (horizontaler Tabulator)
- \v** *Vertical Tab* (vertikaler Tabulator)
- ** *Backslash* (Schrägstrich rückwärts)
- \'** *Apostrophe* (Zeichenkonstanten-Begrenzer)
- \"** *Quotation mark* (Zeichenketten-Begrenzer)
- \?** *Question mark* („Trigraphen“-Anfangszeichen)

Mit **\x** kann auch direkt die äquivalente Hexadezimal-Darstellung eines *ASCII*-Zeichens eingegeben werden, z.B. **\x40** (@).

Zeichen-Ketten (*string-constant*)

Zeichenketten (Strings) belegen immer mindestens 1 Byte. Sie werden immer automatisch mit **\x0** (*ASCII*: NUL) abgeschlossen. An diesem Zeichen wird das Ende des Strings erkannt. Ein leerer String enthält immer die *ASCII*: NUL. Sie werden als *ASCII*-Zeichen eingegeben werden und sind in Anführungszeichen (" ") eingeschlossen. Sie dürfen jedes *ASCII*-Zeichen außer Backslash (\), Apostroph (') oder Zeilenende (*ASCII*: LF) enthalten.

Es dürfen ebenfalls die *Escape-Sequenzen* der Zeichenkonstanten für die 11 Sonderzeichen verwendet werden.

Beispiele für Stringkonstanten:

"Ich bin eine Stringkonstante." /* benötigt 30 Bytes incl. der \0 */
"Ein Backslash \ und Anführungszeichen \" im String."

Aufzählungskonstanten (*enumeration constant*)

Aufzählungskonstanten benutzen das Schlüsselwort **enum** und weisen Namen ganzzahlige Konstante zu. Die Konstanten müssen nicht eindeutig sein.

Beispiel für Aufzählungskonstanten:

enum Semester { E1A, IE1 = 3, D7 = E1A + 3, I5 } ;
/* Hier hat **E1A** den Wert **0**, **IE1** den Wert **3**, **D7** den Wert **3**, **I5** den Wert **4** */

Operatoren in C

Operator	Beispiel	Erläuterung
()	main (void)	Parameterliste, Klammerung
[]	Feld [22]	Index
->	Zeiger -> Mitglied	Auswahl einer Strukturkomponente
.	Jahr.Monat	Auswahl einer Strukturkomponente
!	if (! (a > b)) ..	logisches NOT
~	a = ~b	bitweise NOT (1-Komplement)
++	i++ oder ++i	ähnlich zu: i - i+1
--	i-- oder --i	ähnlich zu: i - i-1
+	a = +b;	plus
-	a = -b;	minus
(typ)	a = (long) b+c;	casting: als long auswerten
*	a = *ptr;	Verweisoperator oder Zeigerdefinition
&	scanf ("%d", &ind);	Adressoperator
sizeof	a = sizeof (long);	liefert Länge eines Typs in Anzahl Bytes
*	a = b * c;	Multiplikation
/	a = b / c;	Division
%	a = b % c;	Rest der Division (modulo)
+	a = b + c;	Addition
-	a = b - c;	Subtraktion
<<	a = b << 3;	Linksschieben (um 3 Bit)
>>	a = b >> 2;	Rechtsschieben (um 2 Bit)
<	if (a < b) ...	kleiner
<=	if (a <= b) ...	kleiner gleich
>=	if (a >= b) ...	größer gleich
>	if (a > b) ...	größer
==	if (a == b) ...	gleich
!=	if (a != b) ...	ungleich
&	a = b & c;	bitweise UND
^	a = b ^ c;	bitweise ANTIVALENZ
	a = b c;	bitweise ODER
&&	if ((...) && (...)) ...	logisches UND
	if ((. .) (. .)) ...	logisches ODER
?:	a = (b > c) ? b : c;	bedingter Ausdruck
=	a = b;	Zuweisung
+= usw.	a += b;	entspricht a = a + b; -= *= /= %= <<= >>= &= ^= =
,	int a, b;	Reihung

Priorität und Bearbeitungsrichtung der Operatoren

C unterscheidet unäre und binäre Operatoren:

Ein unärer Operator bezieht sich nur auf die folgende Variable oder Konstante.

Beispiel: unärer Minusoperator (-): `char a = -7 ;`

Ein binärer Operator verknüpft die vorhergehende Variable oder Konstante mit der folgenden Variablen oder Konstanten.

Beispiel: binärer Minusoperator (-): `c = a - b ;`

Prioritätsgruppe	Operator	Bearbeitungsrichtung
1	<code>() [] -> .</code>	→
2	<code>! ~ ++ -- + - (typ) * & sizeof</code>	←
3	<code>* / %</code> (binär)	→
4	<code>+ -</code> (binär)	→
5	<code><< >></code>	→
6	<code>< <= > ></code>	→
7	<code>== !=</code>	→
8	<code>&</code>	→
9	<code>^</code>	→
10	<code> </code>	→
11	<code>&&</code>	→
12	<code> </code>	→
13	<code>? :</code>	←
14	<code>= += -= /= usw.</code>	←
15	<code>, →</code>	

Die Operatoren sind in der Tabelle in Prioritätsgruppen geordnet.

- Die Priorität der Operatoren in der Prioritätsgruppe n ist jeweils untereinander gleich und höher als die Priorität der Operatoren in der Prioritätsgruppe $n+1$.
- Bei gleicher Priorität gilt die jeweils angegebene Bearbeitungsrichtung. Die Bearbeitungsrichtung gilt für die **Operatoren des C-Quellcodes, nicht für die Anordnung der Operatoren in der oben angegebenen Tabelle!**
- Unär haben die Operatoren `+`, `-` (Vorzeichen) und `*` (Objektorperator) eine höhere Priorität als binär.

Beispiel:

`x = !~a % b++ / --c ;` entspricht: `x = (!(~(-a))) % ((b++) / (--c)) ;`

Kontrollstrukturen in C

Befehl	Beispiel	Anmerkung
while	<pre>while (i > limit) { summe += i; i--; };</pre>	Schleife wird gegebenenfalls nicht ein einziges Mal ausgeführt.
do while	<pre>do { summe += i; i--; } while (i > limit);</pre>	Schleife wird immer mindestens einmal ausgeführt.
for	<pre>for (i = 0 ; i < 5 ; i++) summe += i;</pre>	Schleife wird gegebenenfalls nicht ein einziges Mal ausgeführt.
continue	<pre>while (i != 0) { ... i--; if (i%3 == 0) continue; ... }</pre>	Den gegenwärtigen Schleifendurchlauf abbrechen, Schleife gegebenenfalls aber mit dem nächsten Durchlauf fortsetzen.
break	<pre>while (1) { ... i--; if (i == 0) break; ... }</pre>	– Programmausführung nach der Schleife fortsetzen: Schleife also abbrechen – Einen <i>switch</i> sofort verlassen
if	<pre>if (a > max) max = a;</pre>	Bedingte Ausführung
if else	<pre>if (a > b) max = a; else max = b;</pre>	Alternative Ausführung
switch case	<pre>switch(i) { case 0: printf ("i = 0"); case 1: printf ("i = 0 oder i = 1"); break; case 2: printf ("i = 2"); break; default: printf ("i != 0, 1, 2"); }</pre>	Wahl zwischen mehr als 2 Alternativen

Programmstruktur

Typischer Aufbau eines C-Programms

```
#include <stdio.h>    /* Bibliotheken einbinden */
#include
...

float func1 (int, long); /* Funktionsprotoypen deklarieren */
int func2 (int);
...
int i, j, k;           /* globale Variable definieren */
long m1, m2;
...
int main (void)        /* Hauptprogramm main */
{
    int k;             /* lokale Variable von main definieren */
    ...
    return 0;          /* Rücksprung ins Betriebssystem */
}
...
float func1 (int j, long l3)/* Quelltext der Funktion func1 */
{
    long m1;           /* lokale Variable von func1 definieren */
    int i, n;
    float f = 1.0;      /* ggf. lokale Variable von func1 initialisieren */
    ...
    return f;           /* Ende von func2 , Rücksprung & Rückgabe von i */
}
...

int func2 (int i)      /* Quelltext der Funktion func2 */
{
    long m1;           /* lokale Variable von func2 definieren */
    int lnt;
    ...
    return lnt;         /* Ende von func2, Rücksprung&Rückgabe von lnt */
}
```

Gültigkeitsbereich von Variablen

```
#include <stdio.h>

...
float func1 (int, long);
int func2 (int);

...
int i, j, k;                                /* Diese globalen Variablen sind dem Hauptprogramm
                                           und */ long m1, m2; /* allen Funktionen bekannt,
                                           sofern sie nicht durch lokale
                                           Variable verdeckt sind. */

...
int main (void)
{
    int k;                                  /* Die lokale Variable k verdeckt die globale Variable k
*/
    ...
    return 0;
}

...
float func1 (int j, long m3) /* Die lokalen Variablen j, m1 und i verdecken */
{                               /* die globalen Variablen j, l1 und i. m3 und n sind
    nur in func1 bekannt. */
    long m1;
    int i, n;
    float f = 1.0;
    ...
    return f;
}

...
int func2 (int i)                /* Die lokalen Variablen m2 und i verdecken die
                                globalen Variablen m2 und i. */
{
    long m2;
    int l1;
    ...
    return l1;
}
```

Gültigkeitsbereich und Vereinbarung von Variablen

C bietet folgende Möglichkeiten zur Vereinbarung von Variablen:

- Variable in verschiedenen Funktionen dürfen dieselben Namen haben.
- Namen für Variablen und Prototypen sind dem C-Compiler ab der Stelle bekannt, an welcher der Compiler auf sie trifft.
- Formale Parameter und in Funktionen vereinbarte Variable werden am Ende der Funktion wieder vergessen.
- Namen können verdeckt werden: Durch formale Parameter und in Funktionen vereinbarte Variable werden globale Variable mit demselben Namen abgeschattet.
- Globale (externe) Variable müssen nicht am Anfang stehen.
- Variable dürfen in jeder Verbundanweisung definiert werden.

Beispiel für Variablendefinition und Verdeckung:

```
#include <stdio.h>
int main(void)
{
    short i = 5;
    printf ("i = %d\n", i);
    {
        int i = 10;          /* Die Variable i wird hier neu definiert. Die
        printf ("i = %d\n", i); /* Neudefinition gilt nur für diese Verbundanweisung
    }                       { ... }. Das alte i wird verdeckt*/
    printf ("i = %d\n", i);
    return 0;
}
```

Wert- und Adressübergabe bei Funktionen

Bei der Parameterübergabe an Funktionen unterscheidet man Wertübergabe und Adressübergabe. Bei der Adressübergabe kann das Original des Parameters verändert werden, bei der Wertübergabe nicht.

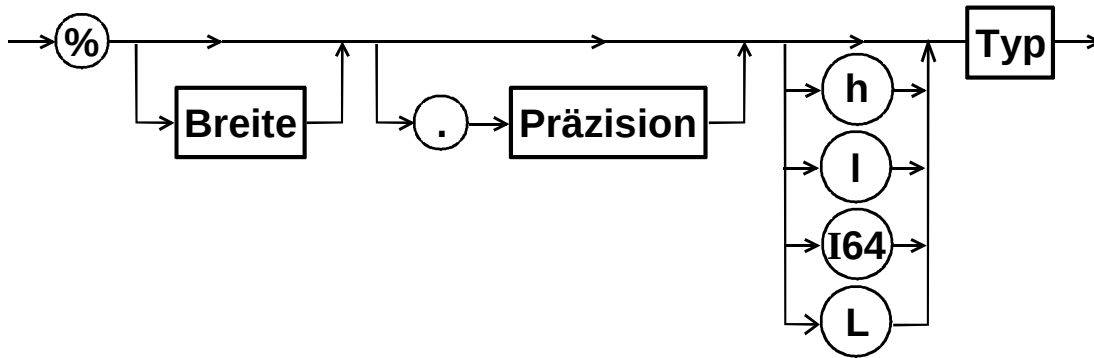
	Wertübergabe Original kann <u>nicht</u> verändert werden	Adressübergabe Original kann verändert werden	
Einfache Variable	<pre>void func (int i) ; { = i ; i = ... ; }</pre> Beispiele zum Aufruf: <pre>int j ; func (7) ; func (j) ;</pre>	<pre>void func (int *p_i) ; { = *p_i ; *p_i = ... ; }</pre> Beispiele zum Aufruf: <pre>int j, *p = &j ; func (&j) ; func (p) ;</pre>	
Feld		1. Möglichkeit: <pre>void func (int f []) ; { = f [0] ; f [3] = ... ; }</pre>	2. Möglichkeit: <pre>void func (int *p_f) ; { = *p_f ; *(p_f + 3) = ... ; }</pre> Beispiele zum Aufruf: <pre>int feld [5], *p_feld = feld; ... func (&feld [0]); func (p_feld);</pre>
Struktur	<pre>typedef struct { int tag, monat, jahr; } DAT ; void func (DAT d) ; { = d.tag ; d.monat = ... ; }</pre>	<pre>typedef struct { int tag, monat, jahr; } DAT ;</pre>	
	Beispiel zum Aufruf: <pre>DAT datum ; func (datum);</pre>	1. Möglichkeit: <pre>void func (DAT *p) ; { = (*p).tag ; (*p).monat = ... ; }</pre>	2. Möglichkeit: <pre>void func (DAT *p) ; { = p->tag ; p->monat = ... ; }</pre> Beispiele zum Aufruf: <pre>DAT datum, *p_d = &datum; func (p_d);</pre>

Eingabe- und Ausgabefunktionen (Bibliothek *stdio.h*)

Prototyp	Erläuterung
<code>int printf (const char *format, Variable1, Variable2, ..) ;</code>	Formatierte Ausgabe auf <i>stdout</i> . Rückgabewert: Anzahl der Variablen; bei Fehler: -1
<code>int fprintf (FILE *fp, const char *format , Variable1, Variable2, ..) ;</code>	Formatierte Ausgabe auf ein Gerät oder in eine Datei. Wie <i>printf</i> , nur ist man nicht auf <i>stdout</i> festgelegt; <i>fprintf</i> (<i>stdout</i> , ..) entspricht <i>printf</i> (..).
<code>int sprintf (char *buffer, const char *format , Variable1, Variable2, ..) ;</code>	Formatierte nullterminierte Ausgabe in <i>buffer</i> . Rückgabewert: Anzahl der ausgegebenen Zeichen (Länge des strings); bei Fehler: -1
<code>int scanf (const char *format, &Variable1, &Variable2, ..) ;</code>	Formatierte Eingabe von <i>stdin</i> . Rückgabewert: Anzahl der korrekt bearbeiteten Eingabefelder
<code>int fscanf (FILE *fp, const char *format , &Variable1, &Variable2, ..) ;</code>	Formatiertes Lesen aus einer Datei oder von einem Gerät; sonst wie <i>scanf</i> .
<code>int sscanf (const char *buffer, const char *format , &Variable1, &Variable2, ..) ;</code>	Der durch <i>buffer</i> definierte Speicherbereich enthält den Eingabestring; sonst wie <i>scanf</i> .
<code>FILE *fopen (const char *filename, const char *mode) ;</code>	Datei mit dem Zeiger <i>*filename</i> auf den Dateinamen öffnen. Der Rückgabewert ist ein Zeiger auf die Struktur <i>FILE</i> . bei Fehler: 0
<code>FILE *feof (FILE *fp) ;</code>	Auf Dateiende testen. Rückgabewert: $\neq 0$ (Dateiende) $= 0$ (kein Dateiende).
<code>FILE *fclose (FILE *fp) ;</code>	Datei mit dem Zeiger <i>*filename</i> auf den Dateinamen schließen. Der durch <i>FILE</i> definierte Puffer wird gelöscht; noch nicht geschriebene Daten werden vorher geschrieben, noch nicht gelesene Daten gelöscht. Rückgabewert: 0 bei Fehler: EOF
<code>int fflush (FILE *fp) ;</code>	Einen Dateipuffer leeren. Ausgabe: gepufferte Daten werden geschrieben. Eingabe: undefiniert. Rückgabewert: 0 bei Fehler: EOF

Eingabe- und Ausgabeformate (1)

- Standardausgabefunktion **printf (..)**:



Breite [optional]: legt die minimale Feldbreite fest:

Wenn die Ausgabe weniger Zeichen erzeugt als die minimale Feldbreite, werden zusätzliche Leerzeichen ausgegeben.

Wenn die Ausgabe mehr Zeichen erzeugt als die minimale Feldbreite, werden die zusätzlichen Zeichen ausgegeben.

Präzision [optional]: legt die Genauigkeit fest.

String (Zeichenkette): maximale Anzahl der auszugebenden Zeichen

Integer (ganze Zahl): minimale Anzahl der Ziffern, bei Bedarf werden führende Nullen ergänzt

Float (Typ f oder e): Anzahl der Ziffern nach dem Dezimalpunkt

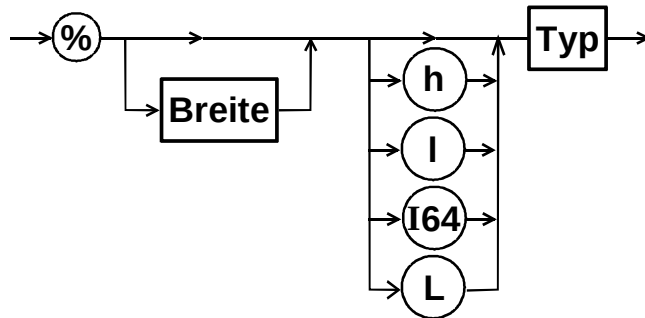
Längenangabe [obligat bei **short**, **long**, **hyper**, **double**, **long double**]: legt den Ausgabedatentyp fest.

Typ	Länge: h, l, L, I64	Ausgabetyt
d, i, u	h	short
	l	long
	I64	hyper
f	l	double
	L	long double

Typ	Datentyp: Ausgabeformat
d	int : Ausgabe dezimal ggf. mit Vorzeichen (-32768 ... 32767)
i	int : wie d
u	int : Ausgabe dezimal ohne Vorzeichen (0 ... 65535)
x	int : Ausgabe hexadezimal ohne Vorzeichen und 0x-Präfix
p	pointer : Ausgabe der Adresse eines Zeigers
f	Gleitpunktzahl : Ausgabe dezimal <u>ohne</u> Exponent: [-]mmmm.dddddd
e	Gleitpunktzahl : Ausgabe dezimal <u>mit</u> Exponent: [-]m.dddddd e ±xxx genau 1 Ziffer vor dem Dezimalpunkt
c	char : einzelnes Zeichen
s	string (Zeichenkette): Ausgabe bis zum <i>ASCII</i> -Zeichen <i>NUL</i> = '\0'

Eingabe- und Ausgabeformate (2)

- **Standardeingabefunktion**
scanf (..):



Breite [optional]: legt den maximale Breite des Eingabefeldes fest.
Ein Eingabefeld wird vorzeitig beendet, wenn bei der Eingabe ein unzulässiges Zeichen auftritt

Längenangabe [[**obligat** bei **short**, **long**, **hyper**, **double**, **long double**]: legt den Eingabedatentyp fest.
Diese Angabe wirkt nur auf numerische Typen. Sie hat z.B. bei den Typen c und s keine Wirkung

Typ	Länge: h, l, L, I64	Zeiger auf
d, i, u	h	short
	l	long
	I64	hyper
f	l	double
	L	long double

Typ: legt den einzugebenden Datentyp fest:

Typ	Datentyp: Eingabeformat
d	Zeiger auf eine vorzeichenbehaftete Festkommazahl (<i>signed decimal integer</i>)
u	Zeiger auf eine vorzeichenlose Festkommazahl (<i>unsigned decimal integer</i>)
i	Zeiger auf eine Festkommazahl: ganze Zahl • dezimal, wenn die erste Ziffer $\neq 0$ ist • oktal, wenn die erste Ziffer = 0 ist • hexadezimal, wenn die ersten w Ziffer = 0x oder 0X sind
f	Zeiger auf eine Gleitpunktzahl, zulässig sind • ein optionales Vorzeichen • eine Ziffernfolge ggf. mit Dezimalpunkt • ein optionaler Exponent (e oder E), ein optionales Vorzeichen und eine ganze Zahl Format: $[\pm]<\text{Vorkomma}>[.<\text{Nachkomma}>][\text{E} \text{e} [\pm<\text{Exponent}>]]$
c	Zeiger auf char : einzelnes Zeichen
s	Zeiger auf string (Zeichenkette): Einlesen bis zum 1. ASCII-Whitespace - Zeichen: (<i>Blank</i> = ' ', <i>NUL</i> = '\0', <i>VT</i> = '\v', <i>HT</i> = '\h', <i>line feed</i> = '\n', <i>form feed</i> = '\f')

Einige mathematische Funktionen der Bibliothek *math.h*

Funktion	Erläuterung
<code>double sin (double x)</code>	Sinus , x im Bogenmaß
<code>double cos (double x)</code>	Kosinus , x im Bogenmaß
<code>double tan (double x)</code>	Tangens , x im Bogenmaß
<code>double asin (double x)</code>	Arcus Sinus , $-1 \leq x \leq +1$, $-\pi/2 \leq asin \leq +\pi/2$
<code>double acos (double x)</code>	Arcus Kosinus , $-1 \leq x \leq +1$, $0 \leq acos \leq +\pi$
<code>double atan (double x)</code>	Arcus Tangens , $-\pi/2 \leq atan \leq +\pi/2$
<code>double atan2 (double y, double x)</code>	Arcus Tangens , $atan(y/x)$ $-\pi \leq atan2 \leq +\pi$
<code>double sinh (double x)</code>	Sinus Hyperbolicus
<code>double cosh (double x)</code>	Kosinus Hyperbolicus
<code>double tanh (double x)</code>	Tangens Hyperbolicus
<code>double exp (double x)</code>	e^x
<code>double log (double x)</code>	natürlicher Logarithmus $\ln(x)$, $x > 0$
<code>double log10 (double x)</code>	dekadischer Logarithmus $\log(x)$, $x > 0$
<code>double pow (double x, double y)</code>	x^y
<code>double sqrt (double x)</code>	$\sqrt{x}$
<code>double ceil (double x)</code>	aufrunden
<code>double floor (double x)</code>	abrunden
<code>double fabs (double x)</code>	$ x $

7-Bit-ASCII-Code (1) / DIN 66003-Code

MS- Stelle →	Hexadezimal	0	1	2	3	4	5	6	7
LS- Stelle ↓									
Hexa- dezimal	Binär →	B ₆ B ₅ B ₄ B ₃ B ₂ B ₁ B ₀							
	↓ B ₃ B ₂ B ₁ B ₀	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0	0 0 0 0	NUL	DLE	SP	0	@	P	`	p
1	0 0 0 1	SOH	DC1	!	1	A	Q	a	q
2	0 0 1 0	STX	DC2	“	2	B	R	b	r
3	0 0 1 1	ETX	DC3	#	3	C	S	c	s
4	0 1 0 0	EOT	DC4	\$	4	D	T	d	t
5	0 1 0 1	ENQ	NAK	%	5	E	U	e	u
6	0 1 1 0	ACK	SYN	&	6	F	V	f	v
7	0 1 1 1	BEL	ETB	‘	7	G	W	g	w
8	1 0 0 0	BS	CAN	(	8	H	X	h	x
9	1 0 0 1	HT	EM	)	9	I	Y	i	y
A	1 0 1 0	LF	SUB	*	:	J	Z	j	z
B	1 0 1 1	VT	ESC	+	;	K	[	Ä	k { ä
C	1 1 0 0	FF	FS	,	<	L	\	Ö	l ö
D	1 1 0 1	CR	GS	-	=	M	]	Ü	m } ü
E	1 1 1 0	SO	RS	.	>	N	^	n	~ ß
F	1 1 1 1	SI	US	/	?	O	_	o	DEL

7-Bit-ASCII-Code (2)

• Bedeutung der Sonderzeichen

ACK	Acknowledge	positive Rückmeldung
BEL	Bell	Klingel
BS	Backspace	Rückwärtsschritt
CAN	Cancel	Ungültig
CR	Carriage Return	zum Anfang derselben Zeile (Wagenrücklauf)
DC	Device Control	Gerätesteuerung
DEL	Delete	Löschen
DLE	Data Link Escape	Datenübertragungsumschaltung
EM	End of Medium	Ende des Aufzeichnungsmediums
ENQ	Enquiry	Stationsaufforderung
EOT	End of Transmission	Ende der Übertragung
ESC	Escape	Code-Umschaltung
ETB	End of Transmission Block	Ende des Übertragungsblocks
ETX	End of Text	Ende des Textes
FF	Form Feed	Seitenvorschub
FS	File Separator	Hauptgruppentrennzeichen
GS	Group Separator	Gruppentrennzeichen
HT	Horizontal Tabulation	Horizontaltabulator
LF	Line Feed	Zeilenvorschub
NAK	Negative Acknowledge	Negative Rückmeldung
NUL	Null (0x0)	Stringendezeichen
RS	Record Separator	Untergruppentrennzeichen
SI	Shift-In	Rückschaltung
SO	Shift-Out	Dauerumschaltung
SOH	Start of Heading	Anfang des Kopfes
SP	Space	Leerzeichen
STX	Start of Text	Anfang des Textes
SUB	Substitute Character	Substitutionszeichen
SYN	Synchronous Idle	Synchronisierung
US	Unit Separator	Teilgruppentrennzeichen
VT	Vertical Tabulation	Vertikaler Tabulator

MS DOS Zeichensatz

Codepage 850 (deutsch, links) bzw. Codepage 437 (englisch, rechts)

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-		☐	☐	♥	♦	♣	♠	.	■	◊	◉	♂	♀	♪	♫	*
1-	▶	◀	↕		¶	§	—	↑	↑	↓	→	←	↶	↷	▲	▼
2-		!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6-	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{		}	~	△
8-	Ç	ù	é	â	ä	å	ä	ç	ê	ë	è	ï	î	ï	Ä	Å
9-	É	æ	Æ	ð	ö	ø	ù	û	ý	Ö	Û	ø þ	£	Ø ¥	× ₣	/
A-	á	í	ó	ú	ñ	Ñ	ª	º	¿	@	¬	½	¼	¿	«	»
B-	☐	☐	☐		†	À	Á	Â	Ë	Ü	¶		¶	¶	¶	¶
C-	⌒	⊥	⊥	⊥	-	+	±	±	±	±	±	±	±	=	±	±
D-	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø	ø
E-	Ó α	ß	Ô Γ	Ô π	ð Σ	Ö σ	μ	þ γ	þ Φ	Ú θ	Û Ω	Ü δ	ý ω	Ÿ ∞	ˉ ∈	˘ ∩
F-	- ≡	±	= ≥	¼ ≤	¶ ∫	§ J	+	, ≈	°	- ·	.	! √	· ·	·	■	