BEZEICHNUNG

printf, fprintf, sprintf, vprintf, vfprintf, vsprintf − formatierte Ausgabe

ÜBERSICHT

#include <stdio.h>

int printf(const char * *format*, **...);**
int fprintf(FILE * *stream*, **const char *** *format*, **...);**
int sprintf(char * *str*, **const char *** *format*, **...);**

#include <stdarg.h>

int vprintf(const char * *format*, **va_list** *ap*);
int vfprintf(FILE * *stream*, **const char *** *format*, **va_list** *ap*);
int vsprintf(char * *str*, **char *** *format*, **va_list** *ap*);

BESCHREIBUNG

Die Funktionenfamilie **printf** erzeugt Ausgaben in einen *format* wie unten beschrieben. **Printf** und **vprintf** schreiben ihre Ausgabe auf *stdout*, dem Standardausgabekanal; **fprintf** und **vfprintf** schreiben in den angegebenen Ausgabekanal *stream*; **sprintf**, und **vsprintf** schreiben in den String *str*.

Diese Funktionen schreiben die Ausgabe unter Kontrolle eines *format*−Strings der angibt, wie die folgenden Argumente (oder Argumente, auf die über **stdarg**(3) zugegriffen wird) für die Ausgabe konvertiert werden.

Diese Funktionen geben die Anzahl der Zeichen zurück, die ausgegeben wurden (ohne abschließendes "\0" zum Terminieren von Strings). **snprintf** und **vsnprintf** schreiben maximal *size* Bytes (inklusive abschließendem '\0'), und geben −1 zurück, wenn die Ausgabe auf dieses Limit gekürzt werden mußte.

Der Format-String setzt sich zusammen aus Null oder mehr Anweisungen: normale Zeichen (nicht %), welche unverändert zum Ausgabekanal kopiert werden; und Umwandlungsspezifikationen, welche jeweils Null oder mehr Argumente fordern. Jede Umwandlungsspezifikation wird durch das Zeichen % eingeleitet. Die Argumente müssen genau zu den Umwandlungsspezifikatoren passen. Nach dem % erscheint das folgende nacheinander:

- Null oder mehr der folgenden Flags:

  # gibt an, daß der Wert in eine "alternative Form" gewandelt werden soll. Bei den Umwandlungen **c**, **d**, **i**, **n**, **p**, **s**, und **u** hat diese Option keine Einfluß. Bei der Umwandlung **o** wird die Genauigkeit der Zahl erhöht um zu erzwingen, daß das erste Zeichen des Ausgabestrings eine Null ist (ausser wenn ein Null-Wert ausgegeben wird mit einer expliziten Genauigkeit von Null). Bei den Umwandlungen **x** und **X** wird einem Ergebnis ungleich Null der String '0x' (oder '0X' bei **X**) vorangestellt. Bei den Umwandlungen **e**, **E**, **f**, **g**, und **G** enthält das Ergebnis immer einen Dezimalpunkt, auch wenn ihm keine Ziffern folgen. (Normalerweise tritt ein Dezimalpunkt nur in Ergebnissen auf, wenn ihm eine Ziffer folgt.) Bei den Umwandlungen **g** und **G** werden nachfolgende Nullen nicht aus dem Ergebnis entfernt, wie sie es normalerweise würden.

  0 Auffüllen mit Nullen. Bei allen Umwandlungen außer **n** wird der umgewandelte Wert links mit Nullen, nicht mit Leerzeichen aufgefüllt. Wenn eine Genauigkeit bei einer numerischen Umwandlung (**d**, **i**, **o**, **u**, **i**, **x**, und **X**), angegeben ist, wird das Flag **0** ignoriert.

  − (ein negatives Feldgrößenflag) zeigt an, daß der umgewandelte Wert linksbündig zur Feldgrenze gesetzt wird. Außer bei der Umwandlung **n** wird der

umgewandelte Wert rechts mit Leerzeichen aufgefüllt statt links mit Nullen. Ein − übersteuert ein **0** falls beide angegeben sind.

´ ´ (ein Leerzeichen) gibt an, daß ein Leerzeichen vor einer positiven Zahl bleiben soll, die durch einen Vorzeichenwechsel entstanden ist. (**d**, **e**, **E**, **f**, **g**, **G**, oder **i**).

\+ gibt an, daß vor alle durch Vorzeichenwechel entstandenen Zahlen das Vorzeichen gesetzt wird. Ein + übersteuert ein Leerzeichen, falls beide angegeben sind. **'** gibt an, daß die Ausgabe bei einem numerischen Argument guppiert werden soll, wenn die lokale Spracherweiterung dieses angibt. Beachte, daß viele Versionen vom **gcc** diese Option nicht parsen kann und stattdessen eine Warnung ausgeben.

- Eine optionale Dezimalzahl, die die minimale Feldlänge angibt. Wenn der umgewandelte Wert weniger Zeichen als die Feldlänge hat, wird er links mit Leerzeichen aufgefüllt (oder rechts, wenn das Flag für Linksbündigkeit gesetzt ist).
- Eine optionale Genauigkeit in der Form eines Punkts ('.') gefolgt von einer optionalen Zahl. Wenn die Zahl weggelassen wird wird eine Genauigkeit von Null angenommen. Dies gibt die minimale Anzahl der Ziffern an, die bei den Umwandlungen **d**, **i**, **o**, **u**, **x**, und **X** erscheinen, bzw. die Anzahl der Ziffern nach dem Dezimalpunkt bei **e**, **E**, und **f** , die maximale Anzahl von signifikanten Ziffern bei **g** und **G** , oder die maximale Anzahl von auszugebenden Zeichen eines Strings bei **s .**
- Das optionale Zeichen **h**, das angibt, daß eine folgende Umwandlung **d**, **i**, **o**, **u**, **x**, oder **X** zu einem Argument *short int* oder *unsigned short int* gehört, oder daß eine folgende Umwandlung **n** zu einem Zeiger auf ein Argument *short int* gehört.
- Das optionale Zeichen **l** (el), das angibt, daß eine folgende Umwandlung **d**, **i**, **o**, **u**, **x**, oder **X** auf einen Zeiger auf ein Argument *long int* oder *unsigned long int* angewendet wird, oder daß die Umwandlung **n** zu einem Zeiger auf ein Argument *long int* gehört. Linux unterstützt eine nicht-ANSI kompatible Benutzung von zwei **l** Flags, die ein Synonym für **q** oer **L** sind. Daher kann **ll** in Verbindung mit Realzahl-Konvertierungen benutzt werden. Von dieser Verwendung wird trotzdem strikt abgeraten.
- Das Zeichen **L**, daß angibt, daß eine folgende Umwandlung **e**, **E**, **f**, **g**, oder **G** zu einem Argument *long double* gehört. Beachte, daß *long long* nicht in *ANSI C* spezifiziert ist und daher nicht portabel für alle Architekturen ist. • Das optionale Zeichen **q**. Dieses is äquivalent zu **L**. Siehe Abschnitte STANDARDS und BUGS für Kommentare zur Benutzung von **ll**, **L**, und **q.**
- Ein Zeichen **Z**, das angibt, daß die folgende Ganzzahl (**d**, **i**, **o**, **u**, **i**, **x**, und **X**) Umwandlung mit einem Argument vom Typ *size_t* zusammenhängt.
- Ein Zeichen, das den Typ der anzuwendenden Umwandlung angibt.

Eine Feldlänge oder Genauigkeit, oder beides, darf durch Einen Stern '*' anstelle einer Zahl angegeben werden. In diesem Fall enthält ein Argument *int* die Feldgröße oder Genauigkeit. Eine negative Feldgröße wird als ein Linksbündigkeitsflag gefolgt von einer positiven Feldgröße aufgefaßt; eine negative Genauigkeit wird behandelt, als wenn sie fehlen würde.

Die Umwandlungsspezifikatoren und ihre Bedeutung:

diouxX
Das Argument *int* (oder eine entsprechende Variante) wird umgewandelt in eine vorzeichenbehaftete Dezimalzahl (**d** und **i**), eine vorzeichenlose oktal- (**o**), Dezimal- (**u**), oder Hexadezimalzahl (**x** und **X**). Die Buchstaben **abcdef** werden für Umwandlungen **x** benutzt; die Buchstaben **ABCDEF** für Umwandlungen **X**. Die Genauigkeit, sofern vorhanden, gibt die minimale Anzahl vor Ziffern an, die Auftreten muß; wenn der umgewandelte Wert weniger Ziffern benötigt wird er links mit Nullen aufgefüllt.

DOU Das Argument *long int* wird in eine vorzeichenbehaftete Dezimalzahl, vorzeichenlose Oktal- oder Dezimalzahl umgewandelt, als wenn das Format **ld**, **lo**, beziehungsweise **lu**

wäre. Diese Umwandlungszeichen werden mißbilligt und werden eventuell verschwinden.

eE Das Argument *double* wird gerundet und umgewandelt in das Format [−]d**.**ddd**e**dd, wobei eine Ziffer vor dem Dezimalpunkt erscheint und die Anzahl der Ziffern dahinter der Genauigkeit entspricht; wenn die Genauigkeit fehlt wird sie als 6 angenommen; wenn die Genauigkeit Null ist erscheint kein Dezimalpunkt. Eine Umwandlung **E** benutzt den Buchstaben **E** (in Gegensatz zu **e**) um den Exponenten einzuleiten. Der Exponent enthält immer mindestens zwei Ziffern; wenn der Wert Null ist ist der Exponent 00.

f Das Argument *double* wird gerundet und umgewandelt in dezimale Notation im Format [-]ddd**.**ddd, wobei die Anzahl der Ziffern hinter dem Dezimalpunkt der Genauigkeit entspricht. Wenn die Genauigkeit fehlt wird sie als 6 angenommen; wenn die Genauigkeit Null ist erscheint kein Dezimalpunkt. Wenn ein Dezimalpunkt erscheint befindet sich mindestens eine Ziffer davor.

g Das Argument *double* wird umgewandelt in das Format **f** oder **e** (oder **E** für die Umwandlung **G**). Die Genauigkeit gibt die Anzahl der signifikanten Stellen an. Wenn die Genauigkeit fehlt werden 6 Ziffern zurückgegeben; wenn die Genauigkeit Null ist wird sie als 1 angenommen. Form **e** wird benutzt wenn der Exponent kleiner als −4 oder größer als oder gleich der Genauigkeit ist. Nachfolgende Nullen im Bruchteil werden entfernt; ein Dezialpunkt erscheint nur wenn er von mindestens einer Ziffer gefolgt wird.

c Das Argument *int* wird umgewandelt in ein *unsigned char*, und das resultierende Zeichen wird ausgegeben.

s Das Argument "*char **" wird erwartet als ein Zeiger auf ein Array vom Typ Character (Zeiger auf einen String). Zeichen auf diesem Array werden bis zu (aber nicht einschliesslich) des terminierenden **NUL**−Zeichens ausgegeben; wenn eine Genauigkeit angegeben ist werden nicht mehr Zeichen als die angegebene Anzahl ausgegeben. Wenn eine Genauigkeit angegeben ist braucht kein Null-Zeichen vorhanden zu sein; wenn die Genauigkeit nicht angegeben ist oder größer als die Array-Größe ist, muß das Array ein beendendes Zeichen **NUL** enthlaten.

p Das Zeiger-Argument "*void **" wird hexadezimal ausgegeben (wie bei **%#x** oder **%#lx**).

n Die Anzahl der bis hierhin ausgegebenen Zeichen wird in dem Integer gespeichert, der durch das Zeiger-Argument "*int **" (bzw. Äquivalent) gegeben ist. Kein Argument wird umgewandelt.

% Ein '%' wird ausgegeben. Kein Argument wird umgewandelt. Die komplette Umwandlungsspezifikation ist '%%'.

In keinem Fall führt eine nicht existierende oder kleine Feldgröße zum Abschneiden des Feldes; wenn das Ergebnis länger als die Feldgröße ist wird das Feld erweitert um das Ergebnis aufzunehmen.

BEISPIELE

Um Datum und Zeit in der Form 'Sunday, July 3, 10:02' auszugeben, wobei *weekday* und *month* Zeiger auf Strings sind:

```
#include <stdio.h>
fprintf(stdout, "%s, %s %d, %.2d:%.2d\n",
        weekday, month, day, hour, min);
```

Um mit fünf Dezimalstellen auszugeben:

```
#include <math.h>
#include <stdio.h>
fprintf(stdout, "pi = %.5f\n", 4 * atan(1.0));
```

Um einen 128 byte - String zu belegen und in diesen zu schreiben:

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

```
#include <stdarg.h>
char *newfmt(const char *fmt, ...)
{
        char *p;
        va_list ap;
        if ((p = malloc(128)) == NULL)
                return (NULL);
        va_start(ap, fmt);
        (void) vsnprintf(p, 128, fmt, ap);
        va_end(ap);
        return (p);
}
```

SIEHE AUCH

printf(1), **scanf**(3).

STANDARDS

Die Funktionen **fprintf**, **printf**, **sprintf**, **vprintf**, **vfprintf**, und **vsprintf** sind konform zu ANSI C3.159-1989 ("ANSI C"). Das **q**−Flag ist die *BSD 4.4*−Notation für *long long*, während **ll** oder die Bedeutung von **L** eine Ganzzahlkonvertierung in der GNU-Notation ist.

Die Linux-Version dieser Funktionen basiert auf der *GNU libio* Bibliothek. Beachten Sie auch die *info*−Dokumentation der *GNU libc (glibc-1.08)* für eine genauere Beschreibung.

BUGS

Einige Fließkommaumwandlungen erzeugen Speicherverluste unter Linux.

Die Umwandlungsformate **%D**, **%O**, und **%U** sind nicht standard und werden nur aus Kompatibilitätsgründen zur Verfügung gestellt. Sie können unter Linux fehlen.

Alle Funktionen sind voll ANSI C3.159-1989 konform, aber bieten die zusätzlichen Flags **q**, **Z** und **'** genauso wie ein zusätzliches Verhalten der Flags **L** und **l**. Letzteres darf als Bug betrachtet werden, da es das Verhalten des Flags wie in ANSI C3.159-1989 definiert verändert.

Der Effekt, das Format **%p** mit Nullen aufzufüllen (entweder durch das Flag **0** oder durch Angabe einer Genauigkeit), und der geringe Effekt (es gibt keinen) des Flags **#** bei den Umwandlungen **%n** und **%p** , sowie andere unsinnige Kombinationen wie **%Ld**, sind nicht standard und sollten vermieden werden.

Einige Kombinationen von Flags, die in *ANSI C* definiert sind, geben keinen Sinn (z.B. **%Ld**). Während sie unter Linux ein wohl-definiertes Verhalten an den Tag legen, muß es bei anderen Architekturen nicht der Fall sein. Daher ist es normalerweise keine Flags zu benutzen, die nicht in *ANSI C* definiert sind, z.B. die Verwendung von **q** anstelle von **L** in Verbindung mit **diouxX** oder **ll**.

Der Gebrauch von **q** ist nicht der gleiche wie in *BSD 4.4* , da er in Realzahl-Konvertierungen gleichbedeutend zu **L** benutzt werden kann.

Da **sprintf** und **vsprintf** einen unendlich langen Sting annehmen muß der Aufrufer aufpassen, nicht den zur Verfügung stehenden Platz zu überschreiten; dies sicherzustellen ist oft nicht möglich.