NAME

printf – write formatted output

SYNOPSIS

printf *format* [*argument* . . .]

AVAILABILITY

SUNWloc

DESCRIPTION

The **printf** command writes formatted operands to the standard output. The *argument* operands are formatted under control of the *format* operand.

OPERANDS

The following operands are supported:

format
A string describing the format to use to write the remaining operands. The *format* operand is used as the *format* string described on the **formats**(5) manual page, with the following exceptions:

- A SPACE character in the format string, in any context other than a flag of a conversion specification, is treated as an ordinary character that is copied to the output.
- A ∆ character in the format string is treated as a ∆ character, not as a SPACE character.
- In addition to the escape sequences described on the **formats**(5) manual page (\\, **\a**, **\b**, **\f**, **\n**, **\r**, **\t**, **\v**), *ddd*, where *ddd* is a one-, two- or three-digit octal number, is written as a byte with the numeric value specified by the octal number.
- The program does not precede or follow output from the **d** or **u** conversion specifications with blank characters not specified by the *format* operand.
- The program does not precede output from the **o** conversion specification with zeros not specified by the *format* operand.
- An additional conversion character, **b**, is supported as follows. The argument is taken to be a string that may contain backslash-escape sequences. The following backslash-escape sequences are supported:
  - the escape sequences listed on the **formats**(5) manual page (\\, **\a**, **\b**, **\f**, **\n**, **\r**, **\t**, **\v**), which are converted to the characters they represent
  - **\0***ddd*, where *ddd* is a zero-, one-, two- or three-digit octal number that is converted to a byte with the numeric value specified by the octal number
  - **\c**, which is written and causes **printf** to ignore any remaining characters in the string operand containing it, any remaining string operands and any additional characters in the *format* operand.

The interpretation of a backslash followed by any other sequence of characters is unspecified.

Bytes from the converted string are written until the end of the string or the number of bytes indicated by the precision specification is reached. If the precision is omitted, it is taken to be infinite, so all bytes up to the end of the converted string are written. For each specification that consumes an argument, the next argument operand is evaluated and converted to the appropriate type for the conversion as specified below. The *format* operand is reused as often as necessary to satisfy the argument operands. Any extra **c** or **s** conversion specifications are evaluated as if a null string argument were supplied; other extra conversion specifications are evaluated as if a zero argument were supplied. If the *format* operand contains no conversion specifications and *argument* operands are present, the results are unspecified. If a character sequence in the *format* operand begins with a **%** character, but does not form a valid conversion specification, the behavior is unspecified.

argument The strings to be written to standard output, under the control of *format*. The *argument* operands are treated as strings if the corresponding conversion character is **b**, **c** or **s**; otherwise, it is evaluated as a C constant, as described by the ISO C standard, with the following extensions:

- A leading plus or minus sign is allowed.
- If the leading character is a single- or double-quote, the value is the numeric value in the underlying codeset of the character following the single- or double-quote.

If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message is written to standard error and the utility does not exit with a zero exit status, but continues processing any remaining operands and writes the value accumulated at the time the error was detected to standard output.

USAGE

Note that this **printf** utility, like the **printf**(3S) function on which it is based, makes no special provision for dealing with multi-byte characters when using the **%c** conversion specification or when a precision is specified in a **%b** or **%s** conversion specification. Applications should be extremely cautious using either of these features when there are multi-byte characters in the character set.

Field widths and precisions cannot be specified as ∗.

For compatibility with previous versions of SunOS 5.*x*, the **$** format specifier is supported for formats containing *only* **%s** specifiers.

The **%b** conversion specification is not part of the ISO C standard; it has been added here as a portable way to process backslash escapes expanded in string operands as provided by the **echo** utility. See also the **USAGE** section of the **echo**(1) manual page for ways to use **printf** as a replacement for all of the traditional versions of the **echo** utility.

If an argument cannot be parsed correctly for the corresponding conversion specification, the **printf** utility reports an error. Thus, overflow and extraneous characters at the end of an argument being used for a numeric conversion are to be reported as errors.

It is not considered an error if an argument operand is not completely used for a **c** or **s** conversion or if a string operand's first or second character is used to get the numeric value of a character.

EXAMPLES

To alert the user and then print and read a series of prompts:

```
printf "\aPlease fill in the following: \nName: "
read name
printf "Phone number: "
read phone
```

To read out a list of right and wrong answers from a file, calculate the percentage correctly, and print them out. The numbers are right-justified and separated by a single tab character. The percentage is written to one decimal place of accuracy:

```
while read right wrong ; do
  percent=$(echo "scale=1;($right*100)/($right+$wrong)" | bc)
  printf "%2d right\t%2d wrong\t(%s%%)\n" \
          $right $wrong $percent
done < database_file
```

The command:

```
printf "%5d%4d\n" 1 21 321 4321 54321
```

produces:

```
    1  21
  3214321
54321   0
```

Note that the *format* operand is used three times to print all of the given strings and that a **0** was supplied by **printf** to satisfy the last **%4d** conversion specification.

The **printf** utility tells the user when conversion errors are detected while producing numeric output; thus, the following results would be expected on an implementation with 32-bit twos-complement integers when **%d** is specified as the *format* operand:

Argument	Standard Output	Diagnostic Output
5a	5	**printf: 5a not completely converted**
9999999999	2147483647	**printf: 9999999999: Results too large**
-9999999999	-2147483648	**printf: -9999999999: Results too large**
ABC	0	**printf: ABC expected numeric value**

Note that the value shown on standard output is what would be expected as the return value from the function **strtol**(3C). A similar correspondence exists between **%u** and **strtoul**(3C), and **%e**, **%f** and **%g** and **strtod**(3C).

In a locale using the ISO/IEC 646:1991 standard as the underlying codeset, the command:

printf "%d\n" 3 +3 -3 \'3 \"+3 "'-3"

produces:

3	Numeric value of constant 3
3	Numeric value of constant 3
–3	Numeric value of constant –3
51	Numeric value of the character '3' in the ISO/IEC 646:1991 standard codeset
43	Numeric value of the character '+' in the ISO/IEC 646:1991 standard codeset
45	Numeric value of the character '–' in the SO/IEC 646:1991 standard codeset

Note that in a locale with multi-byte characters, the value of a character is intended to be the value of the equivalent of the **wchar_t** representation of the character.

If an argument operand cannot be completely converted into an internal value appropriate to the corresponding conversion specification, a diagnostic message is written to standard error and the utility does exit with a zero exit status, but continues processing any remaining operands and writes the value accumulated at the time the error was detected to standard output.

ENVIRONMENT

See **environ**(5) for descriptions of the following environment variables that affect the execution of **printf**: **LC_COLLATE**, **LC_CTYPE**, **LC_MESSAGES**, **LC_TIME**, **TZ**, and **NLSPATH**.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

>0 An error occurred.

SEE ALSO

awk(1), **bc**(1), **echo**(1), **strtod**(3C), **strtol**(3C), **strtoul**(3C), **printf**(3S), **environ**(5), **formats**(5)