Partial Documentation from Ulm's Oberon Library: Print

NAME

Print - formatted output to streams

SYNOPSIS

```
PROCEDURE F(fmt: ARRAY OF CHAR);
PROCEDURE F1(fmt: ARRAY OF CHAR; p1: ARRAY OF BYTE);
PROCEDURE F2(fmt: ARRAY OF CHAR; p1, p2: ARRAY OF BYTE);
PROCEDURE F3(fmt: ARRAY OF CHAR; p1, p2, p3: ARRAY OF BYTE);
PROCEDURE F4(fmt: ARRAY OF CHAR; p1, p2, p3, p4: ARRAY OF BYTE);
PROCEDURE F5(fmt: ARRAY OF CHAR; p1, p2, p3, p4, p5: ARRAY OF BYTE);
PROCEDURE F6(fmt: ARRAY OF CHAR; p1, p2, p3, p4, p5, p6: ARRAY OF BYTE);
PROCEDURE F7(fmt: ARRAY OF CHAR; p1, p2, p3, p4, p5, p6, p7: ARRAY OF BYTE);
PROCEDURE F8(fmt: ARRAY OF CHAR;
             p1, p2, p3, p4, p5, p6, p7, p8: ARRAY OF BYTE);
PROCEDURE F9(fmt: ARRAY OF CHAR;
             p1, p2, p3, p4, p5, p6, p7, p8, p9: ARRAY OF BYTE);

PROCEDURE S(out: Streams.Stream; fmt: ARRAY OF CHAR);
PROCEDURE S1(out: Streams.Stream; fmt: ARRAY OF CHAR; p1: ARRAY OF BYTE);
PROCEDURE S2(out: Streams.Stream; fmt: ARRAY OF CHAR; p1, p2: ARRAY OF BYTE);
PROCEDURE S3(out: Streams.Stream; fmt: ARRAY OF CHAR;
             p1, p2, p3: ARRAY OF BYTE);
PROCEDURE S4(out: Streams.Stream; fmt: ARRAY OF CHAR;
             p1, p2, p3, p4: ARRAY OF BYTE);
PROCEDURE S5(out: Streams.Stream; fmt: ARRAY OF CHAR;
             p1, p2, p3, p4, p5: ARRAY OF BYTE);
PROCEDURE S6(out: Streams.Stream; fmt: ARRAY OF CHAR;
             p1, p2, p3, p4, p5, p6: ARRAY OF BYTE);
PROCEDURE S7(out: Streams.Stream; fmt: ARRAY OF CHAR;
             p1, p2, p3, p4, p5, p6, p7: ARRAY OF BYTE);
PROCEDURE S8(out: Streams.Stream; fmt: ARRAY OF CHAR;
             p1, p2, p3, p4, p5, p6, p7, p8: ARRAY OF BYTE);
PROCEDURE S9(out: Streams.Stream; fmt: ARRAY OF CHAR;
             p1, p2, p3, p4, p5, p6, p7, p8, p9: ARRAY OF BYTE);
```

DESCRIPTION

Print offers formatted printing in *printf(3)* style to *Streams.stdout* (*F* through *F9*) or to *out* (*S* through *S9*). The procedures convert their parameters (the number of parameters determines the procedure name) and instantiate them into the format string *fmt*.

The format string is interpreted as follows: Any character not belonging to an escape sequence introduced by \ or a format element introduced by **%** is simply appended to *Streams.stdout* resp. *out*. Escape sequences are substituted by a single character while format elements are instantiated by the next *p*? parameter.

Format elements must conform to the syntax following:

```
FormatElement = "%" {Flags} [Width] [Scale] Conversion .
Flags = "+" | "0" | "-" | "^" | "\" FillChar .
Width = Number | "*" .
Scale = "." Number .
Number = { DecDigits } .
DecDigits = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .
Conversion = "b" | "c" | "d" | "e" | "f" | "g" | "j" | "o" | "s" | "x" | "y" .
```

Each format element defines a field in the output stream that is by default as wide as necessary to insert the result of a parameter conversion. The field width can be expanded by specifying *Width*. If not given as an explicit *Number* but as ***** *Print* uses the value of the next yet unused parameter (interpreted as an integer) as width indication. Values less or equal than the defaults have no effect. In all other cases the output field is filled up by leading blanks. Padding character and alignment strategy may be altered by use of *Flags*:

+
: Any numeric output will be signed (by default positive values do not get a +sign).

--
: The output will be left aligned within its field. This option has no effect if *width* is omitted.

0
: The output of numeric values is padded with leading zeroes This option implies -- and ^.

^
: The padding characters are inserted before the leading sign and the first digit of a number.

\FillChar
: requires *FillChar* to become the padding character.

On output of real values, *Scale* fixes the number of digits following the decimal point. Other numeric output is not affected while strings are cut to the length given by *Scale* before they are aligned within their output fields. *Print* will use the next yet unused parameter as *scale* indication if ***** is specified.

Since *Print* has no idea about the actual types of the arguments corresponding to its formal parameters, *convchar* is used to determine the conversions to be executed for the next yet unused parameter. *Print* will not accept any other conversion character than those listed and described below. In detail the specifications of *convchar* have the following effect:

x
: hexadecimal output of an integer

o
: octal output of an integer

d
: decimal output of an integer

f
: output of a real number in floating point notation

e

output of a real number in its normalized exponential form

g
: output of a real number in floating point (values with exponents greater or equal to -4 and less than the scale (default: 6)) or exponential notation. Trailing zeroes are suppressed.

c
: output of a single **CHAR**.

s
: output of an **ARRAY OF CHAR** until the first null byte (**0X**) or the high bound of the array is reached.

b
: output of a **BOOLEAN** as text "TRUE" or "FALSE".

y
: output of a **BOOLEAN** as text "yes" or "no".

j
: output of a **BOOLEAN** as text "ja" or "nein".

Note that **o**, **x**, and **d** are legal conversion characters to output any type which has the same size (in bytes) as the expected one. This feature can be used to output an address (integer size presumed). Furthermore these conversion characters may be used to output the ascii-value of a **CHAR**. Vice versa **c** may be used to output a character that is specified by a **SHORTINT**-value.

%% is not interpreted as a format element. A single percent character is output instead.

Any appearance of the following escape sequences in format string *fmt* is substituted as listed:

\n
: newline (line terminator as defined by *StreamDisciplines*)

\r
: carriage return (0DX)

\t
: horizontal tab (09X)

\e
: escape (1BX)

\f
: form feed (0CX)

\b
: backspace (08X)

\&
: bell (07X)

\Q
: double quote (22X)

\q
: quote("’")

\\
: backslash ("\")

\[**0-9A-F**]+
: character specified by [**0-9A-F**]+**X**.
