NAME

printf − formatted print

SYNOPSIS

printf(format, arg_1, ...);
char *format;

DESCRIPTION

Printf converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive argument to *printf.*

Each conversion specification is introduced by the character **%**. Following the **%**, there may be

− an optional minus sign ‘‘−’’ which specifies *left adjustment* of the converted argument in the indicated field;

− an optional digit string specifying a *field width;* if the converted argument has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width;

− an optional period ‘‘**.**’’ which serves to separate the field width from the next digit string;

− an optional digit string *(precision)* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

− a character which indicates the type of conversion to be applied.

The conversion characters and their meanings are

d
o
x The integer argument is converted to decimal, octal, or hexadecimal notation respectively.

u The argument is taken to be an unsigned integer which is converted to decimal and printed (the result will be in the range 0 to 65535).

D
O
X The long integer argument is converted to decimal, octal, or hexadecimal notation respectively.

U The argument is taken to be an unsigned long integer which is converted to decimal and printed (the result will be in the range 0 to 4294967295).

f The argument is converted to decimal notation in the style ‘‘[−]ddd.ddd’’ where the number of d’s after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed. The argument should be *float* or *double.*

e The argument is converted in the style ‘‘[−]d**.**ddd**e**±dd’’ where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced. The argument should be a *float* or *double* quantity.

c The argument character is printed.

s The argument is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.

r The argument is taken to be the address of a *printf* argument list (i.e., a vector of *printf* arguments). The current argument list is discarded, and the new list is used.

The ‘‘r’’ format can be used in the following situation:
‘‘error()’’ is a subroutine which takes *printf* arguments (e.g., error("can’t open %s", file);). The source code for error() is:

```
error(arglist)
{
      printf("%r", &arglist);
      exit(1);
}
```

If no recognizable character appears after the **%**, that character is printed; thus % may be printed by use of the string **%%**. In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by calling *putchar.*

SEE ALSO

putchar (III)

BUGS

Very wide fields (>128 characters) fail.