The function printf () in C

The printf () function in the Standard C library sends formatted output to a stdout.

The printf () function in the Standard C library sends formatted output to a stdout.

Declare the printf () function in C

Here is the declaration for the printf () function in the C programming language:

```
int printf (const char * format, .)
```

The parameter printf () in C

format: This is the string containing the text written to stdout. It may contain formatting tags that can be embedded that are replaced by values ??defined in subsequent additional parameters and formatted as required. Prototype tags of the format **% [flags] [width] [. Precision] [length] specifier** , explained as follows:

Character specifier Character d or i Decimal integer with the sign e Scientific symbol (mantissa / exponent) uses the character e E Scientific notation (mantissa / exponent) using characters E f Actual floating point number Decimal g Short use of% e or% f G Compact use of% E or% fo Octal number signed s Character string u Unsigned decimal number x Hexadecimal integers mark X Unsigned hexadecimal integers (uppercase letters) p Pointer address n Do not print anything% characters

flags Description - Align left into the given field width. The alignment must be the default + Forced to precede the result with a plus or minus (+ or -) even with positive numbers. By default, only negative numbers are preceded by a - (space). If no symbols are written, then a space will be inserted before the value # Used with the specifier o, x or X. The value is preceded by 0, 0x or 0X corresponding to values ??other than 0. Using with e, E and f, it forces the recorded output to acquire a decimal pointer even if not any number followed. By default, if no digit follows, no decimal pointer is written. Use with g or G, the result is the same as e or E but the zeroes at the end do not have to be removed from the left 0 pad (left-pad) of the number with the 0s instead of the spaces

width Description (number) The minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with spaces. The value is not cut even if the result is too large. The width is not specified in the format format string, but as an additional integer value parameter preceding that parameter must be formatted

.precision Description.number For Integer Specifiers (d, i, o, u, x, X) - then Precision specifies the minimum

number of digits to be written. If the recorded value is shorter than this number, the result is padded with 0 at the beginning. Value is not cut even if the result is longer. A precision of 0 means that no character is written to the value 0. For e, E and f specifier: this is the number of digits to be printed after the decimal point. With g and G specifier: this is the maximum number of digits to be printed. With s specifier: this is the minimum number of characters to be printed. By default, all characters are printed until the last null character is encountered. With type c: it has no effect. When no precision is specified, the default is 1. If the period is specified without a clear precision value, 0 is assumed. * Precision is not specified in the format format string, but as an additional raw value parameter precedes that parameter that must be formatted

length Description Parameters are interpreted as a short int or unsigned short int (applied only to integer specifiers: i, d, o, u, x and X) l Parameters are interpreted as a long int or unsigned long int for integer specifier (i, d, o, u, x and X), and as a wide char or wide char string for specifiers are c and s L The parameter is interpreted as a long double (applied only to floating point specifier: e, E, f, g and G)
Additional parameters - Depending on the format format string, this function may have an additional parameter array, each containing a value to be inserted instead of each% -tag specified in the format parameter, if. This number of parameters should be the same number as% -tags that expect a value.

Return value:

If successful, the total number of characters recorded will be returned. If it fails, return a negative number.

For example:

The following program C illustrates the usage of the printf () function in C:

```
#include

int main ()
{
int ch;

for (ch = 75; ch = 100; ch ++)
{
printf ("Gia tri ASCII =% d, Ky tu =% cn", ch, ch);
}

return (0);
}
```

Compiling and running the above C program will result:

```
Family ASCII = 75, Ky tu = K
Family ASCII = 76, Ky tu = L
Family ASCII = 77, Ky tu = M
Family ASCII = 78, Ky tu = N
Blessed ASCII = 79, Ky tu = O
Family ASCII = 80, Ky tu = P
Family ASCII = 81, Kyu = Q
Family ASCII = 82, Ky tu = R
Family ASCII = 83, Ky tu = S
ASCII = 84, Ky tu = T
```

```
Family ASCII = 85, Ky tu = U
Blessed ASCII = 86, Ky tu = V
Family ASCII = 87, Ky tu = W
Family ASCII = 88, Kyu = X
Family ASCII = 89, Kyu = Y
Blessed ASCII = 90, Ky tu = Z
Family ASCII = 91, Ky tu = [
Family ASCII = 92, Ky tu =
Blessed ASCII = 93, Ky tu =]
Blessed ASCII = 94, Ky tu = ^
Family ASCII = 95, Ky tu = _
Family ASCII = 96, Ky tu = `
Family ASCII = 97, Kyu = a
Blessed ASCII = 98, Kyu = b
Family ASCII = 99, Ky tu = c
Family ASCII = 100, Ky tu = d
```