printf is used to O/P data to STDOUT (usually the screen). It has many formatting options which we shall look at in a moment.

printf syntax

This is an example of **printf** in its simplest form.

```
#include <stdio.h>

main()
{
  printf("This text will appear on the screen\n");
}
```

printf is passed one formatting argument. The unusual thing about the example (in my mind) is **\n**, this is actually an escape sequence that signals a new line. Without it, any printf's that follow would O/P to the same line. **printf** also takes extra arguments which are inserted into the format string at locations marked with a **%**.

```
#include <stdio.h>

main()
{
  int number=42;
  printf("The answer is %i\n", number);
}
```

What happens here is the **%i** is seen as a formatting identifier for the next argument (number). In this case an integer is expected.

The following is a list of escape sequences.

\n	Newline
\t	Horizontal Tab
\v	Vertical Tab
\b	Backspace
\r	Carriage Return
\f	Form feed
\a	Audible Alert (bell)
\\	Backslash
\?	Question mark
\'	Single quote
\"	Double quote
\000	Oct - Octal Number
\xhh	Hex Number

printf formatting is controlled by 'format identifiers' which, are shown below in their simplest form.

```
%d %i     Decimal signed integer.
%o        Octal integer.
%x %X     Hex integer.
%u        Unsigned integer.
%c        Character.
%s        String. See below.
```

```
%f        double
%e %E     double.
%g %G     double.
%p        pointer.
%n        Number of characters written by this printf.
          No argument expected.
%%        %. No argument expected.
```

These identifiers actually have up-to 6 parts as shown in the table below. They MUST be used in the order shown.

%	Flags	Min Field width	Period	Precision. Maximum field width	Argument type
Required	Optional	Optional	Optional	Optional	Required

%

The % marks the start and therefore is mandatory.

Flags

The format identifiers can be altered from their default function by applying the following **flags**:

```
-      Left justify.
0      Field is padded with 0's instead of blanks.
+      Sign of number always O/P.
blank  Positive values begin with a blank.
#      Various uses:
       %#o (Octal) 0 prefix inserted.
       %#x (Hex)   0x prefix added to non-zero values.
       %#X (Hex)   0X prefix added to non-zero values.
       %#e         Always show the decimal point.
       %#E         Always show the decimal point.
       %#f         Always show the decimal point.
       %#g         Always show the decimal point trailing
                   zeros not removed.
       %#G         Always show the decimal point trailing
                   zeros not removed.
```

- The flags must follow the %.
- Where it makes sense, more than one flag can be used.

Here are a few more examples.

```
printf(" %-10d \n", number);
printf(" %010d \n", number);
printf(" %-#10x \n", number);
printf(" %#x \n", number);
```

Minimum field width.

By default the width of a field will be the minimum required to hold the data. If you want to increase the field width you can use the following syntax.

```
main()
{
    int number    =  5;
    char *pointer = "little";

    printf("Here is a number-%4d-and a-%10s-word.\n", number, pointer);
}

/*********************************
 *
 *    Program result is:
 *
 *    Here is a number-   5-and a-    little-word.
 *
 *********************************/
```

As you can see, the data is right justified within the field. It can be left justified by using the - flag. A maximum string width can also be specified.

The width can also be given as a variable as shown below.

```
   main()
   {
       int number=5;

       printf("---%*d----\n", 6, number);
   }

   /*********************************
    *
    *    Program result is:
    *
    *    ----     5---
    *
    *********************************/
```

The * is replaced with the supplied **int** to provide the ability to dynamically specify the field width.

Period

If you wish to specify the precision of an argument, it MUST be prefixed with the period.

Precision

The Precision takes different meanings for the different format types.

Float Precision

%8.2f

This says you require a total field of 8 characters, within the 8 characters the last 2 will hold the decimal part.

```
%.2f
```

The example above requests the minimum field width and the last two characters are to hold the decimal part.

Character String Maximum field width

The precision within a string format specifies the maximum field width.

```
%4.8s
```

Specifies a minimum width of 4 and a maximum width of 8 characters. If the string is greater than 8 characters, it will be cropped down to size.

* Precision

As with the 'width' above, the precision does not have to be hard coded, the * symbol can be used and an integer supplied to give its value.

Format Identifiers

The format identifier describes the expected data. The identifier is the character that ends Here is a list of the format identifiers as used in 'printf' ,'sprintf' ,'fprintf' and 'scanf'.

1. Except for '%' and 'n', all the identifiers expect to extract an argument from the **printf** parameter list.
2. All of the parameters should be the value to be inserted. EXCEPT %s, this expects a pointer to be passed.

An example.

```
main()
{
    int number=5;
    char *pointer="little";

    printf("Here is a number %d and a %s word.\n", number, pointer);
}
/*********************************
 *
 *      Program result is:
 *
 *      Here is a number 5 and a little word.
 *
 *********************************/
```