**Printf function**

The function printf enable the programmer to output characters, strings, numbers to the screen. The function scanf enable the programmer to input characters, strings, numbers from the user/keyboard. These functions are called input/output functions or IO function for short. Format and conversion specifier are used interchangeably in this document. The format specifier (enclosed in double quotes) specify how the parameters/variables to be printed in a printf function. In the case of scanf, format specifier dictates how the variables to be assigned with the values the user entered. We will discuss printf function now.

The printf function is defined as

printf ( "format specifier", var1, var2, … ) ;

where format specifier is described later in this section and var1, var2, are the list of variables.

Examples of printf statement are

printf ( "Hello World \n " ) ; // where \n is an end of line specifier.

There is no conversion specifier here, other than Hello World to be printed with an end of line.

Because some of you are coming with a background in Java, the below statement is invalid statement in C

printf ( "Hello " + "World " ) ; // You cannot use + to concatenate string.

But you could do this

printf ( "Hello" );

printf ( " " ) ; // space

printf ( "World") ;

printf ( "\n") ;

Alternatively, we could write the four lines as

printf ( "Hello " ); // space is added at the end

printf ( "World\n") ; // end of line added at the end

You could also write

printf ( "Hello\nWorld\n" ); // This will print Hello, followed by end of line and World in next line

This is also a valid

printf ( "\nHello\n\nWorld\n" ); // How many end of lines are printed ?

Printing character variable values than just static string :

To print character values of a variable, we use %c as in the following line

char myGender = 'F' ; // a single character is stored in a single quote

To print this, we write

printf ( "My gender is %c.\nThank You\n" , myGender ) ;

in the above statement, the output would be

My gender is F.

Thank You

To print integer values, we use %d conversion specifier, like

unsigned int age = 30;

printf ( " My age is %u\n", age ) ; // unsigned int

C language is very loosely typed language, you can pretty much mix types and handle the values provided you know what you are doing. For instance, I could rewrite the above lines using unsigned char as

unsigned char age = 30;

printf ( " My age is %u\n", age ) ; // unsigned char

remember, unsigned char holds values from 0 to 255. Because we are within the range, we could get away with it. we could also write the printf as

printf ( " My age is %d\n", age ) ; // unsigned char using %d without a warning from compiler.

We could also write

char age = 30 ;

But you have to absolutely be careful in mixing chars, short, int.

For now, we learning %c to print characters, %u to print unsigned int, %d to print integers. We will proceed to format specifiers.

**PRINTF FORMAT SPECIFICATION**

So, we have being using printf without getting into the details of this function which provides a lot of formats to output our data. The function takes variable number of parameters and the format specifier is included within the quotes, and it dictates the output or interpret the variable. The specification is

printf ( " % [flags] [width] [.precision] [length] **specifier** ", variable )

the % sign begins the specification.

**Conversion Specifier :**

| To Output | Specifier to use is | Example |
| --- | --- | --- |
| character | %c | printf ( "My gender is %c.Thank You\n" , myGender ) ; |
| Signed decimal or integer | %d or %i | printf ( " %d " , integer\_variable ) ;  printf ( " %i " , integer\_variable ) ; |
| short integer | %hd | printf ( "%hd", short\_type\_variable) |
| unsigned short | %hu | printf ( "%hu", unsigned short\_type\_variable) |
| unsigned integer | %u | printf ( " %u " , unsigned\_variable ) ; |
| octal format | %o | printf ( " %o " , unsigned\_variable ) ; |
| Hexadecimal | %X or %x | unsigned int age = 40;  printf ( "My age=%#X\n", age);  // # means print prefix 0x |
| float | %f or %F | float salary = 34000.60;  printf ( "My Salary=%f\n", salary);  // print the fraction in six decimal places (default)  printf ( "My Salary=%.2F\n", salary);  // print the fraction in two decimal places |
| float | %e , e means base 10 | wt = 823.8992;  prints as 8.238992e+02 , Note the point |
| float | %E, E means base 10 | wt = 823.8992;  prints as 8.238992E+02, Note the point |
| print address | %p | printf ( " %p " , address of variable ) ; |
| print string of character | %s | printf ( " %s " , string variable) ; |

Sample printf statements

unsigned int ui = 345;

int i = -40;

char ch = 48;

printf ( " signed %i \n ", i) ;

printf ( " signed %d \n ", i) ; // signed decimal integer

printf ( " unsigned %u \n ", ui) ;

printf ( " %o \n ", ui) ; // unsigned octal

printf ( " %p \n ", &ui) ; // %p to print the address of a variable

printf ( " %c \n ", ch); // print character

We will discuss the width, flags, precision and length parameters of printf function.

Please note , in the following examples, I am printing the formatted output of a value ( 65000 ) within two vertical bars as markers.

### **WIDTH**

Next, we have Width in the specification printf ( " % [flags] [**width**] [.precision] [length] specifier ", variable )

The number of spaces to be used to print the variable. What if the value of the variable is shorter than this number, blank space will be padded. What if the is longer than this number, then the value is not truncated.

### **FLAGS**

Next, we have Flags in the specification printf ( " % [**flags**] [width] [.precision] [length] specifier ", variable )

| **Symbols** | **What it means** |
| --- | --- |
| - | Left Justify  printf ( "|%-10d| \n ", 65000) ;  // print in 10 spaces, - means left justify, would output  |65000 | |
| + | precede with + or – sign even for positive numbers  printf ( "|%+10d| \n ", 65000);  // add + sign though positive number, would output  | +65000| |
| # | used with X, x or u to put ox or Ox in front,  or put a decimal point in case of floats |
| 0 | prefix (left pad) with zeros instead of spaces  printf ( "|%0+10d| \n ", 65000);  // pad with zero unlike spaces, add + sign, 10 spaces… would output  Output of the above statement  |+000065000| |
| space | insert a space if no sign is to written  printf ( "|%0 10d| \n ", 65000); // there is a space after %0  // would output  | 000065000| |
|  |  |
| Default is right justify | printf ( "|%10d| \n ", 65000);  // Right justify by default, would output  | 65000| |
| What if width provided is less than the value in the variable. The width is ignored. | printf ( "|%5d| \n ", 650000); // contains a space, insert a blank space ..would output  |650000|  because 5 spaces is small to accommodate 650000 , so it prints the whole number without truncating. |

### **.PRECISION**

Next, we have .Precision , printf ( " % [flags] [width] [.precision] [length] specifier ", variable )

Now, about the number itself. How many digits should be written ? .precision guides us with the following, NOTE the DOT prefix

|  | **What this means** |
| --- | --- |
| .number | For integers and unsigned integers, number specifies the number of digits to be written. If the number is bigger than the value in the variable, then zeros are padded. If the number is smaller, the entire number is printed.  For example,  printf ( "|%.7d| \n ", 650000); // 7 after the DOT specifies seven spaces, because 650000 is only six digits, output is padded with zeros  printf ( "|%.3d| \n ", 650000); // 3 after the DOT specifies three spaces, because 650000 is bigger, entire number is printed.  In case of float or double, this number specifies the number of digits to be printed after the DOT |

### **LENGTH**

We then have the length in printf ( " % [flags] [width] [.precision] [length] specifier ", variable ) .

The length (not width) specifies the length of data type. For long and long long data typed variables, just %d is not enough, it will print warning.

long x = 65000;

printf ( "%d \n ", x) ; This will warn us because %d only prints integer data types, not long.

So, we need to provide length specifier.

If we use, the length l option, then the specifier becomes %ld for the printf statement

printf ( "%ld \n ", x ) ; // %ld fixes that.

In the above printf example, we used the combination l and d together, for the compiler to expect a long int variable.

Similarly, we could use h or hh along with d or I option for signed short.

## **SIZEOF OPERATOR:**

We could use z option to print the value returned by sizeof operator. Try this,

printf ( "%d \n ", sizeof(int) ); // printf will warn because sizeof returns size\_t, which is a different data type. To fix it, we can use the z option, like below

printf ( "%zd \n ", sizeof(int) );

### **FLOAT**

| **EXAMPLE** | **OUTPUT** |
| --- | --- |
| float wt = 823.8992;  printf ( "%e", wt);  printf ( "%E", wt); | 8.238992e+02  8.238992E+02 |
|  |  |
| wt = 823 ;  printf ( "%e", wt);  printf ( "%E", wt); | 8.230000e+02  8.230000E+02 |
|  |  |
| wt = 823.89999;  printf ( "%e", wt); // rounding  printf ( "%E", wt); | 8.239000e+02  8.239000E+02 |
|  |  |
| wt = 823.8999901345 ;  printf ( "%f\n", wt) ; // introduces errors  printf ( "%e\n", wt) ; // | 823.899963  8.239000e+02 |
| wt = .000098765432199999999999999;  printf ( "%.9f", wt); // 9 spaces after DOT  printf ( "%.8f", wt); // 8 spaces after DOT  printf ( "%.7f", wt);  printf ( "%.6f", wt);  printf ( "%f", wt); // default to 6 | 0.000098765  0.00009877  0.0000988  0.000099  0.000099 |
| printf ( "%e", wt); | 9.876543e-05 |
|  |  |
| wt=789.56; // print in 15 spaces, 4 after DOT  |%015.4f| // presence of 0 means pad zeros  |%15.4f| // pad with spaces  |%-15.4f| // left justify | |0000000789.5600|  | 789.5600|  |789.5600 | |
|  |  |