Formatted Output: The Forgotten C Facility

K. S. Ooi
Foundation in Science
Faculty of Health and Life Sciences
INTI International University
Persiaran Perdana BBN, Putra Nilai,
71800 Nilai, Negeri Sembilan, Malaysia
E-mail: kuansan.ooi@newinti.edu.my
dr.k.s.ooi@gmail.com

Abstract

Even though input and output facilities are critical in programming, we usually do not spend too much time on them in programming courses. When we teach a programming course to beginners, there are other more pressing topics we must cover, and therefore we do not allocate too much time on input/output facilities, other than using them to perform simple tasks of taking input from keyboard and output the result on the screen. In this article, we focus on formatted output facilities of C. The signatures of *format* in formatted output of C using *printf* can still be found in Python.

Keywords: C, C++, formatted output, printf, format, Python
Date: Dec 26, 2020

1. Formatted Output in C: *printf*

Many programmers who come into contact with C/C++ have used formatted output function, *printf*, countless times. This function appears in the Hello World program [1], which has cult following [2], but yet very few have the slightest idea how to structure outputs, numerical or string result, in a specific format. Obviously: Hello World program has cult following but not *printf*.

The *printf* function takes a format string as its first argument. One of the objects of this string is *conversion specifications* (cs). A cs is specified by % as the start character and conversion character (cc) as the end character. In between these two characters, you will see negative sign, period, and numbers. We will start by showing the formatted output in C using a string “Global Warming is Real!”. To make thing more illuminating, we pad white spaces with ~ characters, and hence our string has become “Global~Warming~is~Real!”.

The first program to print this out is given as follows:

Program 1

```
#include <stdio.h>
#include <stdlib.h>

int main(void){
   char* gw = "Global~Warming~is~Real!";
   printf("Output:%s\n",gw);
   return EXIT_SUCCESS;
}
```

However, when we output a string on the screen, it is rather difficult to see the left and right adjustments, if any. To rectify this, I first print the output to an output string, pad any white space with ^ characters. This is shown in the following Program 2. The program will print ^^Global~Warming~is~Real!^^ on the screen. This is an extra work we undertake to see the whitespaces, by distinguishing the internal ones (~) with the external ones (^). Once you master formatted output, you do not have to do that. A *printf* statement will do.

Program 2

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(void){
   char* st = "  Global~Warming~is~Real!  ";
   char output[50];
   sprintf(output,"%s",st);
   int i = 0;
   while(output[i]){
     if (isspace(output[i])) output[i] = '^';
     i++;
   }
   printf("%s\n",output);
   return EXIT_SUCCESS;
}
```

The length of the string `"Global~Warming~is~Real!"` is 23. We summarize the results in the following table.

Table 1: Conversion specifications on `"Global~Warming~is~Real!"`.

cc	Output	Comment
%s	Global~Warming~is~Real!	The `gw` string, without padding of any sort.
%15s	Global~Warming~is~Real!	Still printing the original `gw` message. The number 15 in this example specifies the *minimum* width. Since the length of `gw`, which is 23, exceeds 15, and so the original `gw` is printed.
%35s	^^^^^^^^^^^^Global~Warming~is~Real!	In this case, you direct the output to at least print 35 characters. Since `gw` has 23 characters, 12 whitespaces are padded *on the left*.
%+35s	^^^^^^^^^^^^Global~Warming~is~Real!	This is same as the previous case. By putting a positive in front of 35, we specifically want the *right adjustment*, and so 12 whitespaces are padded on the left.
%-35s	Global~Warming~is~Real!^^^^^^^^^^^^	The negative sign means you want a *left adjustment*, and so the whitespaces, 12 of them, are padded on the right.
%.15s	Global~Warming~	With a period, you specify that you want to print the first 15 characters of `gw`. In the literature, the number following the period is called *precision*.
%20.15s	^^^^^Global~Warming~	The *width* is specified by the number in front of the period. In this specification, it basically says the width of the output is 20, but output the first 15 characters of

		gw. Since there is no negative sign, it is assumed right adjustment.
%-20.15s	Global~Warming~^^^^^	Similar to the previous case. With the negative sign, we have a left adjustment.

2. Crossing Over to Python 3

All is not lost. The knowledge from Table 1 crosses over to Python 3. Program 3 shows this fact. The square brackets are included in the printing to highlight the whitespaces.

Program 3

Program	Output

`gw = "Global Warming is Real!"`
`print("[%s]" % gw)`
`print("[%15s]" % gw)`
`print("[%35s]" % gw)`
`print("[%+35s]" % gw)`
`print("[%-35s]" % gw)`
`print("[%.15s]" % gw)`
`print("[%20.15s]" % gw)`
`print("[%-20.15s]" % gw)`
	`[Global Warming is Real!]`
	`[Global Warming is Real!]`
	`[            Global Warming is Real!]`
	`[            Global Warming is Real!]`
	`[Global Warming is Real!            ]`
	`[Global Warming ]`
	`[     Global Warming ]`
	`[Global Warming      ]`

Formatted output of integers is trivial, so we skip this. The standard C library stipulated that the default precision of floating and double is 6. The first two experiments in Table 2 below bear this fact out. If the cc is g or G (the general format), *printf* will decide the output, %f or %g/%G; the shorter version will be printed. The shorter version can be defined: the general format will use %g/%G if the exponent is less than -4, or the exponent is greater than or equal to the precision; otherwise, use %f. Again, all this is not lost. Python formatted print matches that of the C's.

Table 2: Formatted output of floating point and double precision numbers. Again, the square brackets are included in the last three printings to highlight the whitespaces.

printf statement	Output	Comment
printf("%f\n",4.283937584);	4.283938	The default precision is assumed, which is 6.
printf("%e\n",7.938475657E-20);	7.938476e-020	The default precision of 6 is assumed.
printf("%g\n",23450900.67);	2.34509e+007	%e is used.
printf("%g\n",2.6895e-3);	0.0026895	%f is used
printf("%G\n",2.6895e-5);	2.6895E-005	%E is used.
printf("[%9.5f]\n", 1.23456789);	[1.23457]	Width is 9, precision is 5, and right-adjusted.
printf("[%-9.5f]\n", 0.123456789);	[0.12346]	Width is 9, precision is 5, and left-adjusted.
printf("[%-9.5e]\n", 0.123456789);	[1.23457e-001]	The precision 5 takes priority. Even though the width 9 is specified, this is deemed as minimum width.
printf("[%-15.5E]\n", 0.123456789);	[1.23457E-001]	Precision 5, width 15, and left-adjusted.

The Python 3 formatted output is given in Program 4.

Program 4

Program	Output

print("%f" % 4.283937584)
print("%e" % 7.938475657E-20)
print("%g" % 23450900.67)
print("%g" % 2.6895e-3)
print("%G" % 2.6895e-5)
print("[%9.5f]" % 1.23456789);
print("[%-9.5f]" % 0.123456789);
print("[%-9.5e]" % 0.123456789);
print("[%-15.5E]" % 0.123456789);
	4.283938
	7.938476e-20
	2.34509e+07
	0.0026895
	2.6895E-05
	[1.23457]
	[0.12346]
	[1.23457e-01]
	[1.23457E-01]

3. Concluding Remarks

The output function of C, *printf*, has an impressive flexibility, which fulfils most of the output formatting needs. Python 3 preserves this C capabilities. In the end, C programmers who are proficient in this aspect of C will find that all is not lost when they move to Python, which is the major language for machine learning, artificial intelligence and data science. Python coders, on the other hand, should master this facility, so that when they move across C, they can readily use this knowledge to format the outputs.

References

1. B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall: Englewood Cliffs, NJ, 1978. Second edition, 1988.

2. ACM "Hello World" project at http://www2.latech.edu/~acm/HelloWorld.shtml (1996 - 2007)

Abbreviations

cs – conversion specification
cc – conversion character