

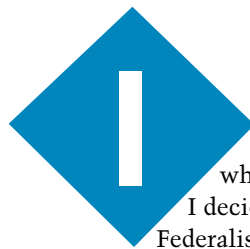
## LESSONS FROM THE TRENCHES

George Martin

### Getting into a Routine

#### Setting Up an Integer Print Routine

As you know, the slightest bump in any routine can cause chaos. George finds himself needing a formatted printing routine, this month, and only perfection will do. The answer to his prayers? A little modification to Turbo C's `printf` and `sprintf`.



Last summer, while on vacation, I decided to read the Federalist Papers. The last presidential election raised so many issues and questions that I was curious about the history behind it all. You should remember from your high school history classes that the United States declared independence from England, signed the Articles of Confederation, and then later signed the Constitution. The Federalist Papers discuss the situation in this country after the Articles of Confederation were written. The papers encouraged the changes proposed in the new Constitution. These papers, written by Hamilton, Madison, and Jay around 1787, were published in a New York newspaper.

I did find what I wanted about the elections and the electoral process from the papers. And, as an added bonus, it gave me a different outlook on the current events. I came away with confidence in the U.S. government. It's fairly dry reading, but I guess I'm accustomed to such reading

from all the technical manuals I get as an embedded engineer.

#### DETERMINING WHAT'S NEEDED

This month, I'll steer clear of trying to write something as complicated as the Federalist Papers and stick to something you can put to use today.

The typical projects that I work on usually need formatted printing. Sometimes, it's just to place output for a debug screen into the proper columns for easier reading. Other times, I need to send an output stream to another piece of equipment, therefore, the format needs to be perfect. I start most projects' coding and desk debug in either Turbo C or Visual C++, and all output goes to the PC's screen or serial port.

Turbo C has several built-in routines for formatted output. Out of these functions, there are two that I use, `printf` and `sprintf`. The prototypes for these are `int printf( const char *format [, argument]... );` and `int sprintf( char *buffer, const char *format [, argument] ... );`.

#### A DEFINITION

Microsoft's Visual C++ V.5.0 description of `sprintf` can be seen in the "Defining Sprintf" sidebar.

These routines are useful when formatting of the output messages is required. The `printf` routine copies characters directly to the `stdout` device, and the `sprintf` routine places the characters into a buffer. Then, you can decide what to do with that buffered data.

#### FOR ARGUMENTS SAKE

For the different types of arguments except floating point types, the math involved to accomplish the formatting is simple and straightforward. But, the floating point requirements of the `printf` routines require that the floating point math routines be linked into the code. Therefore, using the built-in formatted print routines will

expand the code by pulling in floating point routines, even if you never specify a floating point variable or format.

On one recent project, the code expanded by 4 KB when I used *sprintf*. If you need floating point operations in your embedded project, then most of the overhead is already there. But if you don't need floating point and want to use *printf*, then you're stuck with this code expansion. The worst part is that it's code you'll never use.

The solution? I came across a neat download from Mr. Rud Merriman. It's dated 1991 and first appeared in *Embedded Systems Programming*. The code is a reduced version of *printf*, one with all the floating point options removed. This code supports the *l*, *d*, *x*, *s*, and *c* format types along with some special characters like *ln* and *lr*. But, the great part is that you can see Merriman's approach to formatted printing and tailor these rou-

tines to your particular situation.

## THE CODE

The code consists of two C files. The first file is *rprintf.c*, which contains the reduced *printf* routines. The routine that replaces *printf* is called *esp\_printf*. Its output is directed through *outs*, which moves the output to the routine *out\_char*. This *out\_char* routine is the routine that places the characters into the specific location for your hardware. So, your implementation will be different.

The second file is *espmain.c* and is a test suite for the reduced *printf* routine. This is where you can confirm that your modified code behaves properly. It would be foolish not to run these test and not to keep this routine handy to confirm that all is still OK.

I ran this code using Turbo C. In the end, I needed to make two modifications in the *rprintf.c* file, lines 72

and 136. Both lines contain the *const* modifier. The code attempts to modify the variables cast as *const*. I suggest you look up my article, "Everything Changes—Using the Const Modifier" (*Circuit Cellar Online*, October 2000). Basically it's a promise that after a variable is cast as constant you promise not to modify it. I suspect that back in 1991 the compiler that Merriman was using did not have the latest support for the *const* modifier. But today's compilers should have that by now. Try to compile without changing anything and see what happens.

I deleted the *const* modifier in the lines I mentioned earlier, and Turbo C compiled with no errors and the code ran without problems. Next, I'm moving the code, test routines and all, to my target application.

Good luck using these routines. Let me know about any errors, additions,

## DEFINING SPRINTF

### Return value

*sprintf* returns the number of bytes stored in the buffer, not counting the terminating null character.

### Parameters

buffer Storage location for output format, Format-control string argument, Optional arguments

### Remarks

The *sprintf* function formats and stores a series of characters and values in the buffer. Each argument (if any) is converted and output according to the corresponding format specification in the format. The format consists of ordinary characters and has the same form and function as the format argument for *printf*. A null character is appended after the last character written. If copying occurs between strings that overlap, the behavior is undefined.

### Format specification fields: *sprintf*

A format specification, which consists of optional and required fields, has the following form:

**%[flags] [width] [.precision] [{h | l | I64 | L}]type**

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a type character (for example, *%s*). If the percent sign is followed by a character that has no meaning as a format field, the character is copied to

*stdout*. For example, to print a percent sign character, use *%%*.

The optional fields, which appear before the type character, control other aspects of the formatting, such as:

- **type**—Required character that determines whether the associated argument is interpreted as a character, string, or number.
- **flags**—Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.
- **width**—Optional number that specifies the minimum number of characters output.
- **precision**—Optional number that specifies the maximum number of characters printed for all or part of the output field or the minimum number of digits printed for integer values.
- **h | l | I64 | L**—Optional prefixes to type, which specify the size of argument.

or modifications, and I'll publish them. 📄

*George Martin began his career in the aerospace industry in 1969. After five years at a real job, he set out on his own and cofounded a design and manufacturing firm. Typical systems that he designs include servo-motion control, graphical input and output, data acquisition, and remote control. George is a charter member of the Ciarcia Design Works Team and most recently, he's been working on the people-tracking system for Bill Gates' new house. You can reach him at [george.martin@worldnet.att.net](mailto:george.martin@worldnet.att.net)*

## SOFTWARE

The source code is available for download in the Sources section of the article in *Circuit Cellar Online*.

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission.  
For subscription information,  
call (860) 875-2199, or [www.circuitcellar.com](http://www.circuitcellar.com).  
Entire contents copyright ©2001 Circuit Cellar Inc.  
All rights reserved.