Hi guys (and gals),

Reading this collection of comments about Forth certainly brings back warm and nostalgic memories. My name is Henry Laxen, and back in 1983, my friend Mike Perry and I implemented the Forth-83 standard on a bunch of different machines, including the 6800, 8086, 8080 and I don't remember what else.

While it is easy to be derisive about Forth, believe me when I say that there is a lot you can learn about programming, language implementation and and especially "factoring" if you take the time to study the language.

One thing Forth teaches you discipline. Back then, all we had was a 64K (not 64M) address space, and yet we managed to develop some pretty impressive applications. I worked on the Panasonic HHC (the world first hand held computer) which used Forth as its operating system and application language. Inside of a 16K rom we managed to fit not only the Forth language and operating system, but a memory based file system, a calculator, a "memo pad" which allowed you to store messages, an alarm clock/schedular, and the ability to add "capsules" to extend the system. All this ran on a 6502 processsor.

Another great thing about Forth is that a single mind can read and understand the entire Forth system, from low level inner loop to the "meta-compiler" that allows Forth to compile Forth. Unlike C, PERL, (which is my current favorite language), LISP, or God forbid C++, I don't think there is any single human who can comprehend at the bit and byte level how the whole thing works. With Forth you get the whole language and operating system in less than, say 50 pages of source code. Even a feeble mind such as mine can grasp the whole thing. Those of you out there who are planning on making software your vocation will learn many valuable lessons by putting some effort into understand how and why Forth works, even if you never use it in an application.

Another area where Forth "shines" is in debugging hardware. It compiled/interpreted nature make it both very fast and very easy to write simple little hardware probes, that even a hardware engineer can be cajoled into understanding. I remember our hardware guys marvelling at the "bright traces" they could get on the scopes after writing one line of Forth code to probe a particular memory address or io port.

Let me relate to you a few of the "sayings" we used to use back when I was involved in the Forth world.

1. Inside every large program there is a small program crying to get out.
2. In any piece of code larger than 1K, it is always possible to save 1 byte.
3. Never waste a cycle!

This was the nature of Forth, doing more with less. In these days of gigabyte RAM and multi-gigabyte disk, it seems kinda silly, but the nevertheless, the close interaction with the machine, and the understanding gained as to what is really going on when you write a program made learning Forth invaluable.

One last comment. Many people complain that Forth is a write-only language, and that once the code is written no one else can understand it. While this can be true if the programmer is lousy, just the opposite can be true if the programmer is good. I invite any who are interested to inspect the source code for the Forth-83 implementations that Mike and I put together, and I'm sure you will find that with the proper factoring, and the proper choice of names, your can be more readable than lot's of C, LISP, .. etc., code that is out there.

Well, thanks for indulging me in this trip through languages of days past. I will always be grateful for the lessons I've learned by using Forth, even though I find that for web work, PERL is an easier choice these day.

Best wishes, Henry Laxen

Found on Slashdot, 31 Oct 2002