Real men program in C

By Michael Barr
Embedded.com
(08/01/09, 12:00:00 AM EDT)

A couple of months ago, I ate a pleasant lunch with a couple of young entrepreneurs in Baltimore. The two are recent computer science graduates from Johns Hopkins University with a fast-growing consulting business. Their firm specializes in writing software for web-centric databases in a language called Ruby on Rails (a.k.a., "Ruby"). As we discussed many of the similarities and a few of the differences in our respective businesses over lunch, one of the young men made a comment I won't soon forget, "Real men program in C."[1]

Clever though he is, the young man admitted he wasn't making that quote up on the spot. That "real men program in C" is part of a lingo he and his fellow computer science students developed while categorizing the usefulness of the various programming languages available to them. Exploring a bit, I learned the quiche-like phrase assigns both a high difficulty factor to the C language and a certain age group to C programmers. Put simply, C was too hard for programmers of their generation to bother mastering.

Is C a dead language?
For today's computer science students, learning C is like taking an elective class in Latin. But C is anything but history and not at all a dead language. And C remains the dominant language in the fast growing field of embedded software development. **Figure 1** summarizes 13 years of relevant annual survey data collected by the publishers of *Embedded Systems Design*.

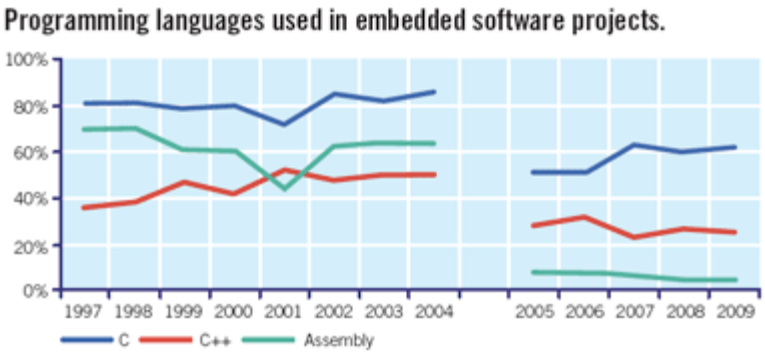

Figure 1

View the full-size image

The discontinuity after 2004 is necessary because the phrasing of the question and permissible answers were changed in 2005. Prior to 2005, the question was phrased, "For your embedded development, which of the following programming languages have you used in the last 12 months?" In 2005, the phrasing became, "My current embedded project is programmed mostly in _____?" Prior to 2005, multiple selections were permitted. This meant that the aggregate data was allowed to sum to over 100% (the average sum was 209%, implying many respondents made two or more selections).

The biggest impact of the survey change from multiple selections to one selection was on the numbers reported for assembly language. Prior to 2005, assembly language was present in an average of 62% of all responses to this question. This should not be surprising, as it is well known that most firmware projects require at least small quantities of assembly code.

After 2004, assembly becomes a minor player--averaging just 7% of all responses across five survey years.[2] This data more closely represents the percentage of projects written mostly or entirely in assembly. The data also show a decline in the popularity of that programming style, from 8% in 2005 to 5% in 2009.

Turning our attention back to C, give Figure 1 a new look with an eye toward the dominance of that language throughout the 13 years of survey data. C was the most used language for embedded software development in the 1997 survey and in the 2009 survey and in every year in between. C has dominated when multiple languages could be chosen (averaging 81%) as clearly as when only the one most-used language could be chosen (averaging 57%).

Remarkably, C appears to have spent the last five years stealing share from assembly as well as from C++. This recent C vs. C++ data defies the expected movement toward ever higher-level languages. C++ is clearly a part of many embedded software projects--and the primary language for about 27% of those coded in the last five years. But my read on the entire 13-year data set is that use of C++ increased rapidly in the late 1990s, peaked in 2001, and has been stable to slightly declining since.[3]

The bottom line is that embedded programmers aren't going to stop using C anytime soon. There are several reasons for this. First, C compilers are available for the vast majority of 8-, 16-, and 32-bit CPUs. Second, C offers just the right mix of low-level and high-level language features for programming at the processor and driver level. Until the use of C starts to turn down in future such surveys, C programming skills will remain important.

Ten billion and counting

Of course, C will not survive as an important programming language if it is widely used by a shrinking subset of programmers. For C to remain important, the number of embedded software developers must not shrink. For better or worse, I believe the opposite is happening. Around 98% of the new CPUs produced each year are embedded. And the number of new CPUs per year is on a long-term upward trend.

Figure 2 shows the rise in the number of new CPUs per year next to the Nasdaq Composite stock index. As anyone can see, the number of new CPUs per year more than doubled in the decade shown. By comparison, the Nasdaq index was down in the same interval. There is a palpable disconnect between the growth in numbers of embedded computers and the prices of technology stocks generally.

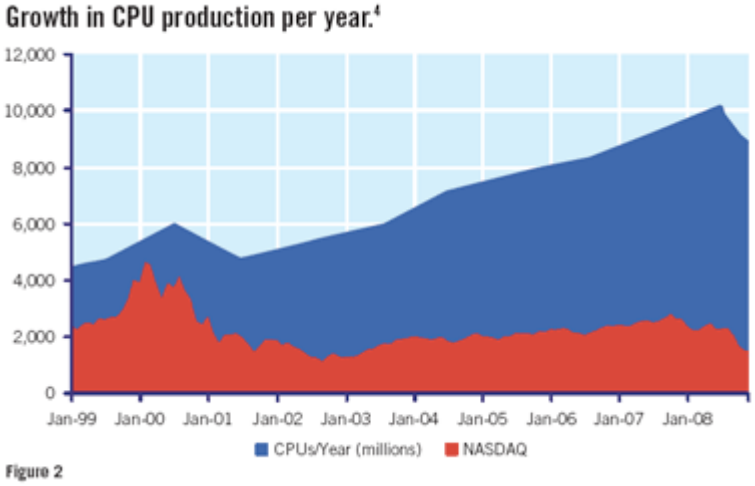

Figure 2

View the full-size image

Based on this data and various other observations over the past 15 years, I conclude that (a) the practice of embedding software into products is on a fast growth curve, and (b) the number of people writing embedded software is growing too. It is important to note that 8-bit processor sales remain a large and growing segment and that these tend to require only one- to two-person programming teams. As processors become cheaper, new applications emerge.

The embedded software education gap

At the same time that C becomes increasingly important to the world, fewer learn how to use that language in school. This is part of a larger "education gap" affecting all organizations that make embedded systems. American institutions of higher learning largely fail to teach the practical skills necessary to design and implement reliable embedded software. From the importance of C's volatile keyword to reentrancy to task prioritization within real-time operating systems to state machine implementation, the trustworthy development processes and firmware architectures for embedded software must be learned on the job.

Figure 3 shows the education gap in a Venn diagram, which is a kind of a tool we mostly did learn in university. Only a little bit of what is studied in electrical engineering curricula is applicable to embedded software development--typically in a lab class requiring assembly programming near the end. And only a little bit more of the applicable stuff is taught in computer science curricula--typically in early classes on computer architecture and now mostly elective courses in C and C++. At least computer science students are taught some of what we must know about software lifecycle management.

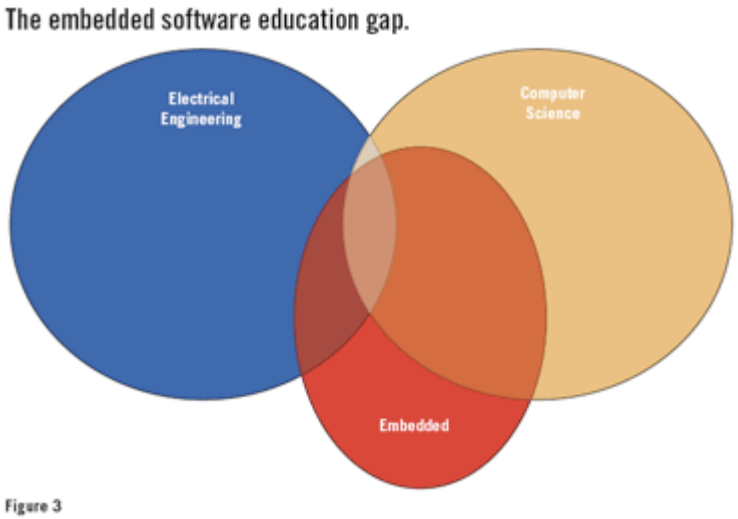

Figure 3

View the full-size image

Though there has been a positive trend toward the addition of computer engineering degrees at many universities, these tend to be too little too late. From what I have seen these CE programs mostly draw courses and professors from the existing EE and CS departments--adding little new content specific to embedded software development. Though the CE program is aimed squarely at educating chip designers and system software developers (a field that includes more than just embedded software), at least one program I am familiar with gives their freshman the choice between a C and a Java track!

Unfortunately, on-the-job learning is also poorly organized in embedded software. It is possible, even common, to start writing firmware with only an EE degree and to begin by making the mistakes of any novice, to receive little code review if any, and to ship a "glitchy" product, only to be rewarded with a new product to work on. Where is the critical feedback that a bug you created frustrated or even injured a user?

Solutions needed

If you accept from the evidence I've presented here that C shall remain important for the foreseeable future and that embedded software is of ever-increasing importance, then you'll begin to see trouble brewing. Although they are smart and talented computer scientists, my younger friends don't know how to competently program in C. And they don't care to learn.

But someone must write the world's ever-increasing quantity of embedded software. New languages could help, but will never be retrofitted onto all the decades-old CPU architectures we'll continue to use for decades to come. As turnover is inevitable, our field needs to attract a younger generation of C programmers.

What is the solution? What will happen if these trends continue to diverge?

Michael Barr *is the author of three books and over 40 articles about embedded systems design, as well as a former editor in chief of Embedded Systems Programming (Embedded Systems Design) magazine. Michael is also a popular speaker at the Embedded Systems Conference and the founder of embedded systems consultancy Netrino. You may reach him at mbarr@netrino.com or read more by him at www.embeddedgurus.net/barr-code.*

Endnotes:

1. I'm sure he wasn't trying to be sexist. Real women surely program in C, too!

2. The 7% average is a pretty strong showing for assembly. By comparison, all other languages averaged less than 5% combined during the same period, including Java (2%) and Basic (1%).

3. The use of Java has never been more than a blip in embedded software development, and peaked during the telecom bubble--in the same year as C++.

4. While the numbers for the Nasdaq index are accurate, the numbers of CPUs per year are my interpolations between a small number of data points of varying reliability.