

## Real men program in C

By Michael Barr

[Embedded.com](http://www.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."<sup>1</sup>

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*.

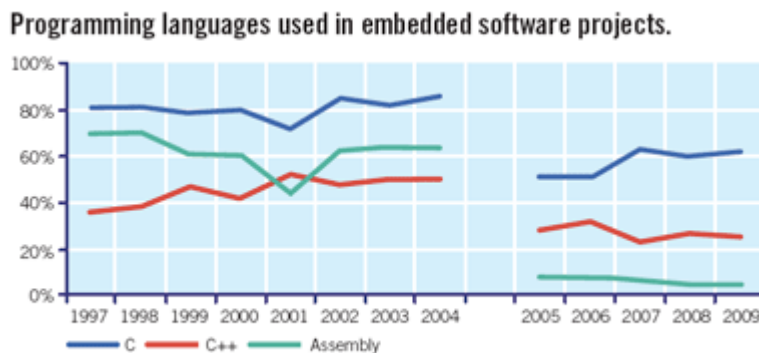


Figure 1

[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.<sup>2</sup> 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%).

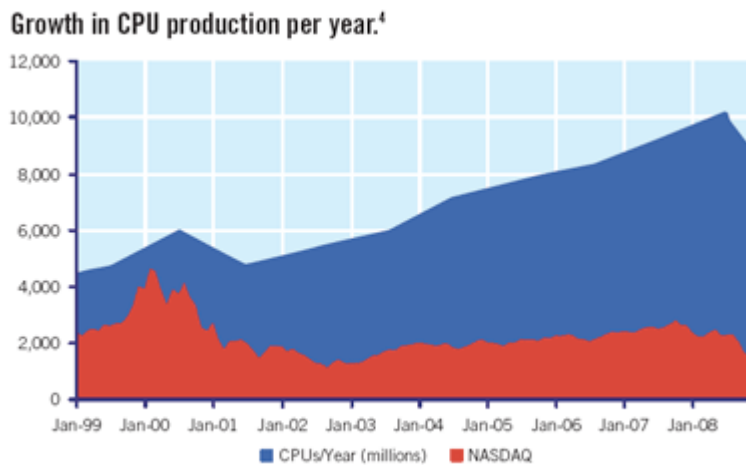
Remarkably, C appears to have spent the last five years stealing share from assembly as well as from C++. This recent C vs. C++ data defies the expected movement toward ever higher-level languages. C++ is clearly a part of many embedded software projects--and the primary language for about 27% of those 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.<sup>3</sup>

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.



[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.

### The embedded software education gap.

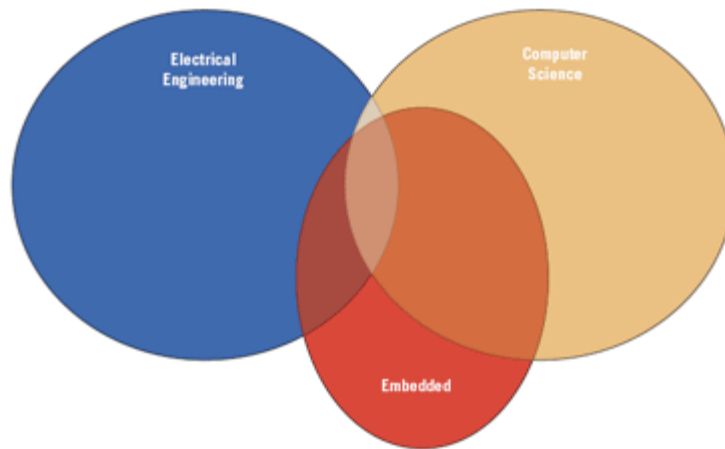


Figure 3

[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](mailto:mbarr@netrino.com) or read more by him at [www.embeddedgurus.net/barr-code](http://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.

#### [Discuss This Article](#)

33 message(s). Last at: Aug 31, 2009 8:34:05 PM

**Caprica****Student**

commented on Aug 2, 2009 6:51:53 AM

Im a second year comp-sci student at the University of Waikato (New Zealand), and I personally love programming in C. Currently looking at linux kernel source code with a hope of being able to make some useful contributions. I have done some assembly in a paper last semester (final assignment was to implement a preemptive multi-tasking kernel in assembly (for a computer designed at the Uni)). Im also planing on buying an embeded computer at some point with the intention of writing some (semi) useful software on in both C and whatever assembly language it takes. Im not sure how to get other students intersted in lower level programming. I guess I just like a challenge and hate Java's verbosity and its over-protectivness it prevents truly learning what the language is doing.

**twomeyw2****Electrical Product Engineer**

commented on Aug 2, 2009 11:19:35 PM

I remember having to program some Intel chips in C while I was in school getting getting my EE degree. We pointed out to the prof that our sole software class (intro to software) used Java. He kind of laughed and said "teach yourselves c, you're EE majors!" It always seemed like there was a rift between the EE profs and the software profs. Some of the EE profs had an elitist complex and refused to even acknowledge software was an "engineering" discipline, it was something beneath them. I don't know if it is like this in other schools but I think this is part of the problem that leads to self taught EEs with varying unconventional styles coming out of school. So, the solution... have the software profs and EE profs sit down for a beer with Obama and some real world embedded coders, maybe you can leave Obama out.

**Alex OD.****Sr Firmware Eng**

commented on Aug 3, 2009 1:36:10 AM

Now I think about it, there was no real EE component to my BSc in CS (with maths). I did have one half unit if Logic Design, but it didn't count to the degree, it was just a extra we did for fun. All I knew about electronics came from high school physics classes.

I did real embedded work for several years before encountering an ADC or DAC, so to me, the rest of the board was just "turn it on, turn it off". It seems all the embedded folks I met were EE majors, or apprenticed electronics guys, I was the oddball.

Offering embedded degrees would not help, as no one would know to take them. Most of us know programming goes with computers. Some know electronics is made up of resistors, capacitors and diodes. But before we get to university we don't know embedded even exists, so how would we know to study it?

At least I know now I am a real man.

**robert.berger*****Embedded Software Specialist - Consulti***

commented on Aug 3, 2009 7:44:56 AM

Hi,

10 years ago real men and women programmed in Assembly now they program in C. I'm, among other things, TA for an embedded systems introductory course in cooperation with CMU. For students who attend this course prior knowledge of Java is considered harmful!

Just for the fun of it you might as well want to have a look at what Linus Torvalds thinks about C and C++: <http://thread.gmane.org/gmane.comp.version-control.git/57643/focus=57918>

Regards,  
Robert

---

Robert Berger  
Embedded Software Specialist

Reliable Embedded Systems  
Consulting Training Engineering  
Tel.: (+30) 697 593 3428  
Fax: (+30) 210 684 7881  
URL: <http://www.reliableembeddedsystems.com>

**DaveArmour*****Sr. Firmware Engineer***

commented on Aug 3, 2009 1:27:47 PM

Being unemployed and looking for a job I am amused at all the "embedded" jobs that require Java, C# and scripting languages. Are they really doing embedded work? To me embedded work requires knowledge of the hardware and how the software interacts with it and assembly and C are the only two languages I have found that do that in a fairly deterministic way. Unless I am missing something, all the rest are just kidding themselves.

**cfbsoftware*****Software developer***

commented on Aug 4, 2009 1:02:46 AM

Maybe that should be "Real old men program in C" 😊

Anybody suffering from the delusion that C is the only viable 'high-level' language for embedded development should have a good hard look at Oberon-07.

<http://www.inf.ethz.ch/personal/wirth/Articles/Oberon.html>

It is as powerful, if not more so, than C but is much less prone to human error.

Chris Burrows  
Development Manager  
CFB Software

Armaide v2.1: ARM Oberon-07 Development System  
<http://www.cfbsoftware.com/armaide.htm>

**NevadaDave*****Embedded Designer***

commented on Aug 4, 2009 11:13:54 AM

I used assembler for pretty much all my Motorola (Freescale, for you young folks), Microchip and Atmel microcontroller projects. Once C compilers with decent IDE's became available, I never used assembler again, and quite honestly, I'm not sure I could at this point! I also use to use BASIC for PC programming, and I've forgotten most of that, too. For "real" embedded design (by "real", I mean designs where it's not a PC in a box, but the hardware is more-or-less invisible to the user) C seems to be a very good blend of high-level abstraction with the low-level bit manipulation. However, I think that the reason C has hung on so long is exemplified (in a sort of backwards fashion) in the post by cfbssoftware: Oberon-07 might be the greatest thing since sliced bread, but if it's not available for the hardware I use, it's a non-player. C is available for pretty much everything nowadays, and those who program for systems that need to be able to manipulate the internal registers & peripherals that modern microcontrollers have will continue to use it for a long time. Just as a side note: the link for the armaide 2.1 should be: <http://www.cfbssoftware.com/armaide/armaide.aspx>

**AlPothoof*****Embedded Software Engineer***

commented on Aug 4, 2009 4:24:25 PM

Figure 1 of this article makes it look like C++ is a minor player in the embedded arena. Has anyone (besides DaveArmour and me) been watching the help wanted ads lately? At least 90% of them are looking for C++ (with some JAVA and C#). And they generally mean 2+ years of professional experience. For RTOS, they generally want Wince (sometimes embedded XP or Vista ???) or embedded Linux.

I think what I see happening is that CS programs are turning out lots of people who know these technologies and the companies figure they can leverage that knowledge by using big, fast chips. They don't care (or maybe don't understand) that none of these technologies are appropriate for high-reliability embedded and/or real-time systems. A classic case of "if the only tool you can find is a hammer, then every problem is a nail."

And, since companies are hiring people with C++ and Windows, that's what the schools are turning out. In this case, if all the problems are nails, all you need to provide are hammers.

Truthfully, I'm starting to feel a little obsolete: I've studied C++ but never used it professionally and, the more I study it, the less I think it appropriate for embedded use. It's not just that it is expensive in terms of resources, it encourages you to do too many risky things. It's bad enough that you can do things like dynamic allocation and recursion in C but, in C++, the language will do it without your ever realizing it, unless you know what its doing behind the scenes (and most people don't; they wouldn't look at the assembly-level output if their project depended on it). And, as Torvald pointed out, if you restrict all the risky behavior, you might better use C and be done with it.

By the way, I still keep my hand in at the assembly-language level. Most of the projects I work with use that for initialization code, if nothing else (although I see some compilers providing that code in C now) and I always do my debugging in mixed-mode so I can tell if the compiler does something unexpected (traditionally, that's been a lack of understanding on my part but I've seen more actual compiler bugs in the past decade than I had in the two decades prior, although that's still only 4 or 5). The only other language I've ever used for real-time systems was Fortran and I haven't used that in about a decade.

**cfbsoftware****Software developer**

commented on Aug 4, 2009 8:53:47 PM

NevadaDave is obviously correct when he says that "if it's not available for the hardware I use, it's a non-player". It is early days yet for Oberon-07 - as far as I know Armaide was the first commercial implementation. However, other activities are already well under way e.g. apart from several different ARM implementations in use, I know of a Russian group working on an implementation for AVR.

I'm not so sure about the emphasis of the comment "those who program for systems that need to be able to manipulate the internal registers & peripherals that modern microcontrollers have will continue to use it for a long time", "it" being C. Microcontroller implementations of Oberon-07 also have equivalent capabilities. Typically they are non-portable microcontroller-specific capabilities so are normally provided in the form of a "SYSTEM" pseudo-module to make it easy to distinguish between portable / non-portable sections of code.

P.S. Thanks for correcting my URL, NevadaDave - how embarrassing 😞

Chris Burrows

CFB Software

<http://www.cfbsoftware.com/armaide>

(without the .htm!)

**Ray Keefe**

**Developer**

commented on Aug 6, 2009 1:39:19 AM

Can see where DaveArmour is coming from because when I think about embedded I usually am thinking small systems tightly coupled to the hardware. Because that is what we mostly do.

However Windows Embedded is another thrust where full on PCs are deployed as Internet Appliances. For these devices C# and other higher level languages make a lot of sense.

And then there are all the Linux and RTOS based systems. The world is changing.

Where I work, we develop embedded systems almost exclusively in C. This covers 8 bit processors like the Atmel AVR and up through 16 bit processors like the MSP430.

The various ARM derivatives are 32bit processors also coded in C though I can see some merit in C++ for these.

Our primary focus is design for low cost electronics manufacture and we execute both the embedded software side and the electronics hardware side. Done this way we can deliver substantial cost savings or improve performance by the choice of what is done in hardware and what is done in software. And of course, how it is done.

The one exception at the moment to coding everything in C is with the Cinterion modules, <http://www.cinterion.com>, using Java Embedded with the Micro Kernel achitecture. These are mobile IP devices where the kernel is probably in C but the application is executed as Java out of FLASH.

In general, executing functions in software saves on electronics manufacturing cost provided it doesn't end up too processor intensive.

I got started in embedded C due to circumstances. I was an analogue hardware engineer and I took a job with a company that needed a C programmer. They made it pretty simple. Either I was able to code in C when I started or the job went to someone else. 2 books and 5 weeks later I was able to do that. There was a lot I didn't know but I was able to start.

Although this was forced upon me I am very appreciative because 21 years later I am a very proficient and productive embedded software developer.

Thanks again for showing the survey data. It seems I'll be employable for the foreseeable future.

Ray Keefe

<http://www.successful.com.au>

**Sundar Srinivasan****Student**

commented on Aug 6, 2009 3:00:26 AM

Great article. C is certainly the language for embedded systems programming. Oberon is great, but still at its beginning stage. The statistics of percentage of embedded developers in C is encouraging, but we have to see how much of those development is really a "new development". Since most of the RTOS and drivers are written in C, modifying it would also contribute to that data. While C is certainly the king of embedded programming languages, we need to look at whether it would be in the future too.

<http://sunnyeves.blogspot.com/>

**ishkabbible****EE**

commented on Aug 6, 2009 3:37:08 AM

Don't be too quick to dismiss very high level languages for embedded work. I design instruments for NASA, and our group codes the firmware in a mixture of C and assembly. But when I'm designing a board, I have to write code in visual basic to glue my various EDA tools together, I build very complex models in several different environments, we distribute the test data through java applets on the team web page, and we analyze the data in either IDL or Igor. Embedded isn't just about the microcontroller. When hiring a software guy, I score about 30% on their C skills, 40% on being able to pick and use the best language to solve today's crisis, and the rest divided between their bit-flipping and communication skills.

**ccorbj****manager**

commented on Aug 6, 2009 12:00:45 PM

From a somewhat specialized perspective (EDA), it is difficult to see C/C++ replaced soon for heavy-lifting applications. There is unquestionably high popularity for languages a level above C (I myself am a Perl fan and would hate to be forced into C unless absolutely necessary). But I observe there is considerable fragmentation above the C/C++ level - Perl, Ruby, Python, Java, ... Java is perhaps the most established of these meta-languages, and already making inroads in handset applications, and I could foresee a day when JREs are successfully implemented directly in hardware (attempt have already been made, but I am unaware of any being wild successes)

While each of these meta-languages has value, I think a dominant standard has to emerge, with equivalent performance across a broad range of applications for C/C++ to be superseded

**K1200LT Rider****Embedded SW Engineer**

commented on Aug 6, 2009 12:06:44 PM

ishkabbible,

Do you work in Florida's Space Coast area? Just curious since that's where I am.

Brad

**ishkabbible****EE**

commented on Aug 6, 2009 2:31:02 PM

Brad,

Nope - I just go there to integrate instruments and watch them launch. NASA (wisely, IMHO) chose a bunch of Arizona desert rats to go to Mars and find the water.

**RussMorash****Software Developer**

commented on Aug 10, 2009 1:03:16 PM

I can also see where DaveArmour is coming from but the days of an embedded application being programmed, shipped and never heard from again are coming to an end. Be it changing settings or calibrating what equals nicely browned on that toaster more and more embedded applications need to have some communication with the outside world. For me that means also having to know and use higher level languages like C# a Java (among others) to easily (as compared to a pc app in C) develop a GUI to interact with the device.

**Susan McCord****software engineer**

commented on Aug 10, 2009 4:09:24 PM

I think part of the reason a younger generation of software engineers is not attracted to embedded software (and hence C) is the lack of money/prestige. I have been stunned to see some job postings lately that require years of embedded software expertise/medical device control expertise/etc. and the pay rate is \$30/hr! If a software engineer coming out of school can go to Google et. al. and get a hefty salary, a very cool reputation and stock options, why are they going to go into a more meticulously demanding job that requires embedded design skills? Companies that need embedded software engineers need to value them appropriately and make these jobs more attractive.

**NevadaDave****Embedded Designer**

commented on Aug 10, 2009 4:31:27 PM

It appears my earlier comment got lost, so if this is just reiteration, please forgive me!

My comment was that we might want to consider adding a new term to the embedded universe, something like, "Complex Embedded" to describe systems or applications that require (or uses) processors or other sufficiently complex logic that require or are best programmed by higher-level, perhaps OO languages. For example - my Atmel 8-bit designs are all done in C, not because it's the only language available for the Atmel AVR's (assembler & Forth are others of which I'm aware) but because it's readily available, has a high-enough level of abstraction to make many functions easy to write, but enough hardware manipulation to handle the basic peripherals of the device. When I wrote a Windows GUI-based configuration application for one of our electronic advance ignition systems, I used C#, because it was a good fit for that kind of system. I think that we have reached the point where we need to start differentiating the complexity/memory/cpu aspects of embedded systems before asking the question about what language we use in writing the software. New terms, anyone?

**perth1415**

**Firmware Lead engineer**

commented on Aug 10, 2009 5:13:12 PM

Interesting article!

I've some points to make -

1. How did you arrive at the Figure-3? Could you provide the sources?

2. "Unfortunately, on-the-job learning is also poorly organized in embedded software."

I strongly believe this point. When I joined the embedded domain as a fresher, I was thrown in to this domain, asked to write some drivers in VxWorks; then I moved into other RTOS's & finally into Firmware without any OS, mainly 802.11 MAC firmware. I never got a chance to work on Linux. I know I could have (& should have) learned a lot about the Linux kernel if I had the zeal. But somewhere I missed it. Recently during a job interview I got a feedback that meant something like this - "9 yrs of experience & you don't know Linux kernel?? Man .. you're a piece of you-know-what!". So, I've lately installed Linux on my PC & started writing a driver from scratch.

I wish somebody advised me to have a peek inside Linux kernel, back when I started in this field.

3. "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."

I don't quite agree with this. As time has progressed, customers are demanding higher throughput, better stability. Based on the hardware design, you (as a software developer) are expected to deliver a certain amount of performance in your driver/firmware. If you can't do that, you lose your job to someone who can. This barely leaves any scope for "not caring to learn" your language - C.

Also, going by your statement, code written some 10 years back were better than code written in new products running in the market now? No offense meant but I don't quite agree to that.

**LukeTeyssier*****Design Engineer***

commented on Aug 11, 2009 8:29:58 AM

Michael Barr raises a good point. Teaching Java/Python/Ruby/TCL, etc. does not teach computer engineering. They may be useful for many tasks, but they let the student avoid really understanding what the computer is doing. They teach how to get something working quickly by trading off speed and size. In this day and age when nearly every technical professional can program (Astronomers/Astrophysicists, Structural Engineers, Civil Engineers, Mathematicians, Chemists, Biologists, Cryptographers, etc.) computer professionals need to do better than hack out code in "some high level language" to earn their daily bread. As a profession all about the tool (computer) they need to be experts in the tool and able to write the foundation that makes efficient use possible for others. They also need to be able to go as deep as necessary when things go badly with all those nice abstractions. So, if you aren't a deep expert in computers then how are you in any way more than just an assistant to those professionals who have a deep knowledge of their own expert field, plus can do your job too? IMHO, if you can't explain in detail what happens from `printf("Hello, world\n")`, all the way down to "fetch, decode, execute", then you either need to find it out, or accept the fact that your job may vaporize as soon as someone figures out how to encapsulate your limited knowledge into a GUI builder or wizard. Remember all those folks with degrees in English and Art getting \$150K a year in 1998 doing html coding? Remember how they talked about the "old" economy and how they were part of the "new wave of computer scientists"? Remember how they got canned in 2001 as soon as decent web design software came out and the economy took a dip? Get ready to update your resume.

I phone screened 110 "Senior Embedded Software Engineers" to get 5-6 who could competently answer "A 'C' Test: The 0x10 Best Questions for Would-be Embedded Programmers". Note that it shouldn't be possible to graduate from a reputable CS curriculum without being able to answer these questions.

**D\_Lundin**

***R&D Manager***

commented on Aug 11, 2009 11:09:06 AM

People don't use C because it's a good language, people use C because it is the least evil option. For embedded realtime systems you really only have the choices listed: C, C++ or assembler.

Assembler will of course be used at some extent "inline", but to write whole programs in it is not an option if you want portable code. At the escalating pace in which the silicon industry plops out new processors, one can almost be certain that the program will live longer than the processor. Portable code is a must in the embedded world, therefore we need high-level languages. Private encapsulation is another good reason why.

I can come up with a few advantages of C++ over C in embedded systems, namely stronger type enforcement, exception handling, standardized inline asm, standardized inlining (C does not have this because nobody but the ISO C committee programs in C99).

But the disadvantages weight heavier in my opinion:

Horrible, unnecessarily complex syntax. C++ is perhaps the least readable programming language of them all. Compare the syntax to Java or C# and you will grimace at the C++ syntax.

Potential for very inefficient programs through inheritance, templates, STL etc. STL classes will enforce a heap when we don't need/want one.

Utterly few fully compliant C++ compilers for embedded systems exist. Many claim they follow the standard, but they almost certainly forgot to implement one detail or another (who can blame them, there are so many unnecessary features). But because of this, C++ programs are very hard to port. Wasn't portability why we wanted a high-level language in the first place?

Object orientation is no good argument. The only part of object orientation that is a must-have in embedded systems is private encapsulation.

C supports this nicely enough with static variables declared at file scope. Initialization through constructors is not something you desire, in fact you will want to avoid this for security reasons and for reduced bootup time.

Also, the most moronic ingredients of C, like for example macros and the way integers are implemented, are not fixed in C++. So stick with C til something better comes up. Maybe some day there will be a deterministic language with fixed integer sizes.

**Jayakumar****CTO**

commented on Aug 12, 2009 1:56:22 AM

Bit funny but it is true. Often I love to program in C when I start a new algorithm development project. Few days might go on how to create data flow and kind of things and start coding in Scilab instead of C. When I code in Scilab, I love to keep C like code flow such that my fellow friends does not have problem when they take my Scilab script and move on to C. Some one ask why do I do this kind of long way journey?. Yes often world is not the one we like to have. For example, C is good language and gives option to write efficient code that can run on multiple hardware platforms. But look at headache it provides to fellow developer. To start with it does puts brake on fast thinking brain when they want move along algorithm development with more concentration on data flow rather then issues to keep track of things in C language. Another one which is very painful headache for Algorithm developers is that till today there does not exist good development environment IDE in which Data flow debugging can be done with ease by using graphs and other associated mechanisms. In Scilab, we don't have the above two headaches ( not fully zero but tending to zero) and makes algorithm development person very efficient in terms of time and obtained result. Thus, in my view we need brand New Tool that can provide Scilab like easy and C like efficiency. Shall we fund a new company to create such a tool?. May be we need to name that as AlgoC.

**WJT****Sr. Sw Eng**

commented on Aug 13, 2009 6:49:23 PM

Real men only program in two languages and they both begin with A, Assembler and Ada. Why don't you see more non-military systems developed in those languages? Cause there aren't enough real men out there to go around.

**TPS****Engineer**

commented on Aug 17, 2009 8:15:24 AM

I see a Bell Curve forming.

**Dave2112****Technical Director**

commented on Aug 17, 2009 5:54:46 PM

I'm saddened to see an article effectively saying programming in C is no place for a woman (clearly she should stick to the home!) With a bit of though the title could have read "Real People Program in C" and the phrase 'That "real men program in C" is part of a lingo he and his fellow computer science students developed' should have read:  
'That "real men program in C" is part of a sexist lingo he and his fellow computer science students developed to keep women out of the profession'

**ESD editorial staff: SRambo**

**ESD managing editor**

commented on Aug 18, 2009 12:54:03 PM

The "real men" isn't intentionally sexist. It's tongue-in-cheek. Ever heard of 1982 book "Real Men Don't Eat Quiche" ([http://en.wikipedia.org/wiki/Real\\_Men\\_Don%27t\\_Eat\\_Quiche](http://en.wikipedia.org/wiki/Real_Men_Don%27t_Eat_Quiche)) or the famous article "Real Men Don't Use Pascal" (<http://practical-tech.com/entertainment/real-men-dont-use-pascal/>)?

**jlcsa**

**Automation Engineer**

commented on Aug 18, 2009 1:17:09 PM

Dear Mr. Barr, I am an embedded software developer for the Canadian Space Agency. C is our language of choice. I am not sad to see Assembly go, but it will always remain useful in certain situations. The only reason we won't go for C++ is as D\_Lundin mentioned: lack of fully-compliant compilers.

Just wanted to say I appreciated reading your article.

**Bill E**

**Senior Bug Exterminator**

commented on Aug 20, 2009 8:54:16 PM

I disagree that "macros .. are not fixed in C++." Template functions take care of the vast majority of cases where one would be forced to use a macro in C, in both a more typesafe manner and without the dreaded namespace pollution of macros.

As for efficiency, you generally don't pay for what you don't use - although you may need to tell the compiler to not generate code for exception handling and/or RTTI if you truly can't afford them. That said, I've found that it's the programmer, not the implementation language that drives an application's ultimate efficiency.

I'm curious about the reference to security as a potential downside of constructors. On the flip side, I'd argue that C++ features such as constructors can increase the safety of code see [http://hissa.nist.gov/sw\\_develop/ir5769/ir5769.1.html](http://hissa.nist.gov/sw_develop/ir5769/ir5769.1.html), to ensure that variables are initialized to a reasonable state. And C++'s destructors allow one to implement RAII, virtually eliminating the chance of resource leaks - here I'm referring to not just memory leaks, but "leaks" of mutexes as well.

**D\_Lundin*****R&D Manager***

commented on Aug 21, 2009 8:35:57 AM

Regarding macros:

While there are better ways to solve things in C and C++, macros are still allowed. The only reason to use macros in ISO C 9899:1990 is portable inlining of functions, a minor concern as modern compilers handle this automatically.

In C99 and C++ I can't think of a single reason to use them. Yet people use them a lot, because they don't know better. There is a tendency of thinking "the language allows it so it can't be bad", or worse: "the language allows it so I'm gonna use some alien, impossible-to-interpret syntax to show off my syntax knowledge".

Regarding constructors:

In safety-critical systems, it is considered poor practice to rely on initialization values. This isn't because of the language, but because of the unreliable nature of RAM. In such systems, there could be days, weeks or years from the point of initialization to the point where the variable is used. If you rely on the initialization values, you leave the safety in the hands of the RAM manufacturer. If you instead leave the initialization constants in NVM as long as possible and set variables in runtime, before they are used, you get increased safety.

Also, in typical safety-critical systems you have no memory leaks at all, as dynamic allocation and heap usage is completely banned. Mutex leaks aren't too common either, as threads/processes tend to exist during the whole program execution. Anything getting created & deleted dynamically is a potential hazard.

In non-critical systems, there are typically no policies for the above and programmers are left to goof around with the RAM as they see fit. Hence the need of various design patterns, smart pointers etc in such systems. I would whole-heartedly agree that constructors/destructors will increase safety drastically in any system allowing dynamic resources.

**Dan\_Saks*****Contributing Editor***

commented on Aug 21, 2009 10:14:14 PM

As I have written in the past, I believe C++ is generally preferable to C for most applications, embedded or otherwise. I believe that some of the remarks in this thread recommending C over C++ are mistaken. See the details in my latest column at <http://www.embedded.com/columns/programmingpointers/219401085>

**DonDiegoDelaVega****Senior Embedded Software Engineer**

commented on Aug 27, 2009 11:30:42 AM

I remember when the phrase was "Real men program in assembler"! In my experience embedded applications fall into one of two categories, timely real-time and time critical real-time, the difference being 100s of Msecs response time to 100s of Nsecs response time. There should be no question that using OOA/OOD is the standard operating procedure used for program design, even for embedded systems. In most of today's embedded designs we have the luxury of megabytes and gigahertz verse the kilobytes and megahertz available 15 years ago in our embedded targets. Given that, I say that C++ would be my choice for timely real-time applications and C for time critical real-time apps. This is because the designer not the compiler has complete control over fixed verse dynamic memory allocation and other things that go on under the hood of C++. A couple a years ago I was told by a Wind River FE that they once implemented VxWorks in C++ but had to replace it with a C version for timing reasons. I strongly believe that a disciplined designer can use OOA/OOD in their C designs (i.e. a collection of procedures that perform a function or service (class) in a single module, constructors, destructors, etc.). I admit that C may not be the only language for time critical real-time apps but currently has the most development environment support for the widest range of target systems. If you check out the leading tool vendors such as Wind River, TI and Matlab, I think you will see what I mean.

**CNutt****CEngine Developer**

commented on Aug 31, 2009 5:03:14 PM

My CEngine is a new kind of processor that runs C code directly. The parser is written in C# and produces the loads for multiple memory blocks that personalize a small FPGA design to run the code. The objective is to minimize the number of clock cycles so that the maximum function can be done in C rather than HDL. The architecture is not the typical von Neumann nor Harvard Bus so parallel operations overcome the effects of pipeline stalls. The point that C is still alive is very encouraging. I am an old EE, computer guy, self taught C/C++ programmer.

I will appreciate any questions/comments, email  
kws15@verizon.net

**S.T.**

**Software Engineer**

commented on Aug 31, 2009 8:34:05 PM

Paraphrasing somebody's comment on Reddit I would say: "Real men produce good and complete system analysis before starting to program with the most appropriate tool."

A good system analysis makes abstraction of the tools used to implement the system. OOD/OOA are not part of this phase but of the next step which is software architecture and design. If the analysis suggests an important degree of repeatability then an OO approach may be chosen. If not SADT or similar techniques will do it.

Good examples are: WIDOWS OS, structural approach in C. MSFC, OOA/OOD in C++.

Any other approach which bypasses the system-analysis and jumps directly to low-level software design will produce unpredictable results and cannot be cited as reference.

IMHO the C/C++ controversy is just another "religious war" without any real support. Ultimately the common-sense and the market decide. Just as an exercise for all OOA/OOD fans out there: please look up and let me know one popular, widely used OS based on this technology, an OS being the perfect example of an embedded system.

Please [login or register here](#) to post a comment or to get an email when other comments are made on this article

### [Track This Thread](#)

Sends you email alerts when someone comments on this article