Our recent [Embedded Revolution](http://beta.electronicdesign.com/embedded-revolution/developers-are-chasing-ai-open-source-and-security-technologies?code=UM_NX7UMB1) survey reinforced the view of C and C++ dominance with embedded programmers, but the results also highlight the differences between the embedded space and others with respect to other programming languages in use. Assembler would not even be mentioned by other developers, plus MATLAB and LabVIEW would probably not hit the top 10.

I touch on this and other results ranging for artificial intelligence and machine learning to safety issues in my white paper, [“Developers are Chasing AI, Open-Source, and Security Technologies.”](http://www.electronicdesign.com/embedded-revolution/developers-are-chasing-ai-open-source-and-security-technologies?code=UM_NX7UMB1)

I thought it might be interesting to hear from some of those people who developed some of the other popular platforms highlighted in these results, since languages like C [have changed over time](http://www.electronicdesign.com/dev-tools/what-s-difference-between-c-now-and-then). Since programming languages are not designed in isolation, I asked what other languages influence them, when they encountered C, and what they are working on now. So, here are some of the responses.

**Bjarne Stroustrup, Creator of C++**

My first programming language was Algol60. I ran it on a Danish GIER computer as part of my first computer science course in 1970. I first tried C in 1973 when I was one of the first four UNIX users in Europe, but I didn’t use it for anything real until I came to Bell Labs in 1979. BC—Before C—is really a very long time ago! Then, I used a lot of languages: assembler and micro-assembler for really low-level stuff; BCPL and Coral, for systems programming; and Algol68 and Simula for more general programing. In addition, I played with more than a dozen other languages. It was easier to pick up a language then because their supporting libraries and tool chains were far simpler.

Algol 60 was the first language (that I know of) that aimed at static type safety. Simula extended that idea to allow you to define your own types. I found that extensibility essential—the few types a language designer or a design committee can think up simply cannot match the varied needs of programmers. C++ takes that idea of static type safety and extensibility and brings it to systems programming and other areas where resources (time, memory, battery life, etc.) are limited. My ideas of resource management came from operating systems, rather than languages. Constructors and destructors are the language embodiment of those ideas.

Currently, I try to improve C++ to make it more flexible while improving static type safety. Eventually, we will reach perfect type- and resource-safety without sacrificing expressiveness, conciseness, or the ability to manipulate hardware. This is an ambitious and noble goal. The [Core Guidelines project](https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md) is my main effort in that direction. We will get there relatively soon.

**James Gosling, Creator of Java**

My first programming language was Focal5 and PDP8 assembler, about 1969. I then moved on to Fortran, PL/1, Pascal, Simula, and CDC assembler. My first encounter with C was in 1976. I am now working at Liquid Robotics on distributed control system for autonomous ocean-going robots.

The programming language had the most impact on me was Simula. It was the first object-oriented language. Really lovely.

**Jeffrey Kodosky, Father of LabVIEW**

The first language I learned and programmed in was Fortran for a class on numerical methods I took as a freshman. I also learned a number of assembly languages over the years, and my two favorites were PDP-8 and, most especially, PDP-11. I thought it was the epitome of elegance.

I encountered C in the mid-’70s.  My first reaction was that the syntax seemed fairly complicated—for instance, all the various assignment operators.  As I learned and used it more I came to appreciate the syntax being a concise way to express more precise semantics.  I’ve enjoyed using C ever since.

Today, I’m still working on LabVIEW development, mostly in C++, although most of the programming I do is actually using LabVIEW—i.e., programming graphically using G.

The language that had the most impact on me was Lisp.  I encountered it in the early ’70s and became fascinated by it. That’s when I realized there was much more to CS than just numeric computation. In fact, it was the catalyst that caused me to switch my graduate studies from theoretical physics to computer science.  As a fun diversion, I wrote and published a Lisp interpreter for the PDP-11 (in assembly language), based on the MIT “blue-book” Lisp which contained a Lisp interpreter written in Lisp.  That ultimately had a profound effect on my thinking as I developed LabVIEW.

**Dr. Benjamin Brosgol, Distinguished reviewer of the original Ada language specification and a member of the design team for Ada 95**

My first language was Fortran. I encountered C in the early 1970s, when it first came out. I still occasionally use C and some Python, but Ada 2012 is what I use for most of my work now. I also want to mention SPARK—that is, Ada adding formal methods.

There were a number of programming languages that had an impact on me. Of course there is Fortran (a person always remembers their first love), since it introduced me to programming. Early on there was Algol 60. It illustrated the sad principle that a language’s technical merit did not imply commercial popularity.

I liked Pascal for its simplicity. My first experience with object-oriented programming was courtesy of Simula. Modula’s enforcement of information hiding and data abstraction was important, as well. Ada has always done an amazing job of integrating high-level features like encapsulation, strong typing, generic templates, tasking, real-time support, and low-level control.

Other languages that deserve honorable mention include LISP for functional programming, SNOBOL for text processing and pattern matching; Java, which helped prevent C++ from taking over the world; and finally, Python, which made it fun to program (at least for me).

**William Wong, Technology Editor, *Electronic Design***

Since I put together this article, I’ll add my own two cents. I actually started with BASIC on a minicomputer and racked up a range of languages early on, including assembler, FORTRAN, and COBOL. I hit quite a few before running into C while I was working at Burroughs, including Algol, ESPOL (Burrough’s systems variation of Algol), Snobol, and APL. I used PL/1 and JCL when I was working at the Center for Disease Control massaging data for SAS. I used Lisp for some research at school and even wrote some apps using Borland’s Prolog.

I hit C when I was working on DEC machines and eventually wound up using it on an Atari ST, as well. Early on the C compilers were not great in terms of optimization, so doing nasty things like assignments in equations was one thing I don’t recommend now.

C, C++, and C# have been in my mix lately because those tend to be the tools available for embedded development platforms I have been evaluating, although I tend not to do production work in this space. Robotics and CMS systems are more common work for me when I am not writing. PHP, Python, and Java have been the languages I have used most for these projects. I even have a few Drupal modules I have shared.

I have used dozens of languages and found each has provided me useful tools and insights. I have tried to grok Haskell and functional programming but have yet to write anything substantial in it. Lately I have been using Ada and trying to get a good handle on SPARK. They have turned out to be quite handy on microcontrollers, and I would recommend those to embedded developers—although beware the myths (see “11 Myths About Ada” on electronicdesign.com).

https://www.electronicdesign.com/technologies/embedded-revolution/article/21805077/before-c-what-did-you-use

call management system – система управления вызовами

encapsulation = encapsulating – инкапсуляция – сокрытие внутренней структуры данных и реализации методов объекта от остальной программы, т. е. включение в объект всей необходимой информации, таким образом, чтобы другим объектам не требовалось знание его внутренней структуры. Доступен только интерфейс объекта, через который осуществляется всё взаимодействие с ним.

strong typing – строгий контроль типов, строгая типизация – в языках программирования - требование, чтобы для каждой переменной был чётко определён тип данных, который может быть ей присвоен; это предотвращает ошибки, связанные с типами данных (type error), во время исполнения программы