C, the Enduring Legacy of Dennis Ritchie

A tribute to the late Dennis Ritchie delivered at Dennis Ritchie Day at Bell Labs, Murray Hill, NJ, September 7, 2012

We have gathered here today to pay tribute to the memory of Dennis Ritchie and to the far-reaching contributions that he has made to society with his software.

Almost everyone has heard the word "software," but relatively few people understand what software really is, what it does, and how much of it there is. In my introductory computer science courses at Columbia, I begin by asking students the question, *How much software is used by the world today? Take all of the software systems used by industry, governments, academia, and society at large. Don't count duplicate systems more than once. How many source lines of code do you think were written to create all these systems?*

After a while, some brave student puts up his hand and says "one million." Then another student says, "No, I think it's a bit more than that – maybe ten million." Then I mention that the Microsoft Windows XP operating system alone is over 45 million source lines of code, and that I saw an article a few years ago that estimated the SAP business application environment has over 250 million lines of code.

After a few more examples of large software systems, I finally say that no one has a precise answer to this question, but I would estimate the world depends on at least one trillion lines of source code. My reasoning goes something like this: assume that there are about 5 million programmers in the world and that the productivity of an average industrial programmer is somewhere between 2,500 and 5,000 lines of tested, documented code per year. So every year, tens of billions of new lines of code are created. Since programmers have been writing software for many decades, we can easily get to a total of many hundreds of billions of lines of source code. And since we have great difficulty jettisoning old software, the legacy base just keeps getting bigger and bigger.

Not only is society at large not aware of what software is and how much of it there is, society is even more unaware of its great software creators. As an experiment, I googled the name "Dennis MacAlistair Ritchie" and got 97,000 hits. For comparison, I googled "Justin Drew Bieber" and got over 20 million hits.

To get a glimpse of the enormity of the impact that Dennis's contributions have made to society, I will just focus on his work on C and Unix. First, let's recap a little bit of history. The language C was created by Dennis in the early 1970s as part of his work with Ken Thompson on developing the Unix operating system. Once C compilers became available, C rapidly became the language of choice for creating software on Unix.

Within a few years C developed a life of its own as a general purpose language even outside of the Unix system. For example, Microsoft implemented its Windows operating system using C. In fact, many mission-critical software systems in the world today are written in C. You can't make a telephone call, fly in an airplane, or drive a car today without using Dennis's inventions. One of the most striking testaments to the enduring legacy of C was contained in a note I just received from Gerard Holzmann who now works at NASA. Gerard said that NASA's Curiosity Rover that just landed in the Gale Crater on Mars with its spectacular sky-crane descent maneuver is controlled by 3.8 million lines of ISO-standard C code. So it took just 40 years for Dennis's invention to go from the Unix room at Bell Labs to the Gale Crater on Mars.

Another measure of Dennis's impact on software is the number of important new programming languages created after C that are derivatives of C or that have been strongly influenced by C.

About a decade after Dennis invented C, Bjarne Stroustrup joined Bell Labs and created the C++ language by adding classes and object orientation to C. I think it is safe to say that many of the most widely used software systems in the world today are written in C or C++, or a combination of these two languages.

But C++ was not the only programming language that was influenced by C. The other very popular languages of today – C#, Java, JavaScript, Objective-C, and PHP – were also strongly shaped by C. In fact, I would venture to say that almost all of the major software systems used by people in the world today have been written in C or languages influenced by C.

I might add that my own research and teaching interests in programming languages and compilers have been heavily influenced by Dennis. As one example, in 1977 Brian Kernighan, Peter Weinberger, and I developed a data-processing language that got to be known as Awk. Awk programs are sequences of pattern-action statements – the patterns are boolean combinations of regular expressions and numbers, and the actions are C-like statements.

Two other domain-specific languages had been created earlier at Bell Labs with a similar pattern-action paradigm. In the mid 1970s, Michael Lesk and Eric Schmidt had developed the Lex language for creating tools to analyze text and for building lexical analyzers for the front ends of compilers. At about the same time Steve Johnson devised the Yacc language for creating parsers for compilers and other language translators. Both Lex and Yacc use C statements to specify semantic actions, and even after several decades these two tools are still the mainstay of today's compilers courses.

In my own programming languages and compilers course at Columbia I have students work in teams of five where each team has to design and implement a new innovative programming language. The teams produce interpreters for their languages using Lex and Yacc. Since it is easy to produce a working compiler with these tools, the students can focus on the creative aspects of language design, rather spending their time implementing a more mundane language without compiler tools.

An example of one of the more interesting languages produced in my compilers course last semester is a language called W2W, for what to wear. A W2W programmer enters into a database the clothes she has in her closet. She then writes a W2W program describing her fashion style. The compiled W2W program reads the weather report for the next day and suggests outfits for the person to wear. Because the focus of the course is on language design, this course has become very popular at Columbia, attracting over 100 students each semester.

At the end of the course, students need to write a project report that includes a section entitled "Lessons Learned." One of the most telling comments ever included in this section was made by a student who wrote "During this course we realized how naive and overambitious we were, and we gained a newfound respect for the work and good decisions that went into languages like C which we've taken for granted for years."

Let's turn our attention now to operating systems. The Unix system that was codeveloped by Dennis Ritchie and Ken Thompson has been as influential in the operating systems world as C has been in the programming languages community. In June 1972 Dennis and Ken were quoted as saying, "... the number of Unix installations has grown to 10, with more expected." Today there are hundreds of millions of Unix systems in use around the world and Unix is a multibillion-dollar-a-year industry. The major non-Microsoft operating systems in use today such as Google's Android, Apple's OS X, and the countless versions of Linux and BSD, all have some kind of Unix at their cores. It is fair to say that today's global information networking infrastructure would not work without the systems that have been derived from Dennis and Ken's original Unix.

One might ask why did Unix and C have so much influence. It is not because there weren't any competing systems or languages at that time. In fact, the first version of Unix was created by Ken

in 1969 as a consequence of him being asked to stop working on the Multics operating system that was being developed jointly by Bell Labs, GE, and MIT in the 1960s. At the time C was growing up, the DOD launched a major initiative to create a general purpose programming language christened ADA for all of its software developers to use. ADA flowered for a while and is still in use today but did not end up being as hardy or as ubiquitous as C.

So why did Unix and C become so successful? This question has been studied a lot and I think there are many reasons for their success. One important reason is the enlightened management style that existed in Bell Labs Research that allowed scientists to pursue promising long-term research efforts with little micromanagement. Another reason, and in my opinion the single most significant reason, is the good taste that Dennis and Ken had for software. When software people encountered or read about Unix and C for the first time, they had an “aha” experience – they said “yes, that’s exactly how an operating system and a systems programming language should be done.” And since the initial implementations were small and readily available, they immediately started using Unix and C, and teaching their colleagues and students how to use Unix and C. A third reason is that Unix was born in the forge of the Unix room where a small number of extremely talented researchers worked together with Dennis and Ken, and contributed ideas and critiques and most importantly used the Unix system itself to create software as it was being developed.

It is not surprising that the popularity and success of Unix and C frustrated some other systems researchers and attracted some envy. In 1989 Richard Gabriel wrote an essay with the oxymoronic title “Worse is Better” trying to explain why the Unix/C approach with its emphasis on simplicity and flexibility was trumping in the marketplace the MIT approach with its emphasis on consistency and completeness. Gabriel’s “Worse is Better” oxymoron was sometimes called “NJ style.” In NJ we preferred to say “Small is beautiful.”

When I first joined Bell Labs in 1967, Richard Hamming, the inventor of Hamming codes, came into my office in the first week I was there and said, “Al, in order to be a great scientist you not only have to do good work, you also have to teach others how to use your work.” I think the early availability of good tutorials and manuals is another key reason why Unix and C became so successful. At the insistence of Doug McIlroy, the early Unix systems had on-line manual pages making the system much easier to learn and use.

And then there is K&R. On top of being a creative software developer, Dennis was a superb technical writer who could explain complex software ideas simply, clearly, and often eloquently to beginners. His book “The C Programming Language” written with Brian Kernighan is in my opinion the finest textbook not only on a programming language but also on programming in general. It is known in the field simply as K&R. In addition to being the initial definition of the C programming language, it taught beginners how to write elegant, useful C programs -- all in under 300 pages.

My programming languages and compilers course that I mentioned earlier has benefitted greatly from K&R. As part of the language implementation effort, I ask each team of students to write a tutorial for their language based on Chapter 1 of K&R and a language reference manual patterned after Appendix A. Once I started insisting that the teams create tutorials and language reference manuals before they start implementing their languages, never has a team failed to deliver a working compiler by the end of the semester.

Dennis and Ken’s far-reaching contributions to programming languages and operating systems have been rewarded with the highest honors a computer scientist can earn. In 1983 Dennis and Ken won the ACM Turing Award, the most prestigious award for technical achievement in computer science. In 1990 they were awarded the National Medal of Technology by President Bill Clinton, and in 2011 the Japan Prize for Information and Communications.

I asked a fellow faculty member at Columbia if he could think of any new computer science development that could potentially have the same impact as Unix and C have had on computing. He said no, the only new thing he could think of that would have the same widespread impact would be a comet hitting the earth.

I would like to conclude my tribute to Dennis by saying a few words about Dennis the man. I found Dennis to be a very private, humble individual. But whenever I talked with him, he was always a peach of a person: caring, broad in his interests, and gracious in his behavior.

When Bell Labs opened a research lab in Beijing, China in the year 2000, Dennis and I went on a speaking tour to a few Chinese universities. At Peking University in Beijing, Dennis gave a talk on Unix and C to an overflow audience of enthusiastic students who treated him like a rock star. At the end of his talk, a perhaps over-enthusiastic student asked Dennis: *Well a long time ago, you did Unix and C. What have you done since then?* Without malice, Dennis replied: *You know, this is the first time I've been asked that question. I'll have to think about the answer.*

Dennis was not averse to expressing his opinions forcefully and candidly. In 1994 a book called the Unix-Haters Handbook was written that screeched Gabriel's Worse is Better philosophy. Dennis wrote an anti-forward for this book which was actually included in the back of the book.

Dennis's epilogue ends: *Here is my metaphor: your book is a pudding stuffed with apposite observations, many well-conceived. Like excrement, it contains enough undigested nuggets of nutrition to sustain life for some. But it is not a tasty pie: it reeks too much of contempt and of envy. Bon appetit!*

Dennis also had a deliciously wry sense of humor. He was once asked: *In your experience, how long does it take for a novice programmer to become a reasonably proficient C developer capable of writing nontrivial production code?*

Dennis replied: *I don't know. I never had to learn C.*

To the software world, Dennis was a genius. To those who were privileged to know him, he was a mensch.

Dennis, we all miss you.

Alfred V. Aho
Lawrence Gussman Professor
Department of Computer Science
Columbia University