Once upon a time...

Once upon a time, in the early 1970's, there was a guy named Dennis Ritchie, who worked at AT&T Bell Labs. He had nothing better to do but develop operating systems and computer languages. The guy was a genius. Anyway, one day (in the early 1970's) he brought forth a high-level computer language that we now know as "C". (It was originally called "B", but was changed to "C". Please don't ask me why, because I don't know.)

By the way, I plan to explain what a "high-level language" is in one of the next sections.

C did not become very popular until the late 1970's because C compilers were not available until then. (I also plan to explain what a "compiler" is, in one of the next sections.)

C programs originally ran on UNIX based systems. UNIX (an operating system) will also be discussed in one of the next sections, was also developed in the early 1970's at AT&T Bell Labs.

Lots of C Compilers were developed and were not completely alike. This became a problem because a program that was written (and compiled to machine code) on one system, would not work on another system. So, if I wrote a checkbook balancing program, for example, and developed in on my microcomputer (PC or Mac) at home, and then tried to run it on a Sun Workstation (for example) running UNIX; that very same checkbook balancing program would not work on the UNIX-based machine. This is not very good as far as program *portability* is concerned. It is important to be able to port (copy, move) programs from one machine to another.

Another analogy (getting away from computers) is if you asked a bar tender to make you a *gin and tonic* , and you got a drink with 5 ounces of tonic, and 1 ounce of gin. Then you went to a different bar tender and asked for another *gin and tonic* (expecting to get 5 ounces of tonic and 1 ounce of gin) and this time got a drink with 4 ounces of tonic and 2 ounces of gin. Well. Well. This is not very standardized, now is it? You'd be upset right? Right? RIGHT??? Yes. We expect certain standards to be followed in life. The same holds true in the computer world.

So, in order to rectify the problem of non-standard compilers, the American National Standards Institute (ANSI) stepped in and formed a committee in 1983 to standardize C compilers, so that people' s code would work on different computers. With an ANSI (pronounced ann-see) definition of C, you are assured that anyone who sells a true ANSI C Compiler has followed the standard. This is one of C's strengths. That it is a *highly portable* language. (More on this later.) So now, if you write a program on one computer (MS Windows based, for example), you can compile and then run it on another (VAX VMS, for example)! What joy!!! Could life get any better than this!?!?!?

It is because of its portability that C is widely used in industry. Another reason C is such a desirable language is because its instructions set (the C commands that we use to write our programs) allows the programmer to manipulate hardware directly (such as memory). This is similar to the *low-level* programming of assembly language; however, we're using a high-level language. Because of this capability, C offers an enormous amount of power and flexibility. (If this made no sense at all to you, don't worry. It will later. :-))

The C language resembles Pascal, in that it is a *highly structured* language. That is, both the code and the data are strongly organized (structured).

Examples of structured code include

- Sequence Structures (code runs in sequence)
- Repetition Structures (for, do, and while loops)
- Selection Structures (if, else, else if, switch)

Examples of structured data include:

- Arrays
- Structs (records)

The C language has some features that no other language has, making it more powerful than all other high-level languages. These features include:

- Pointers
- Operations on Bits.

Both of the above features are quite advanced, and are not covered in this *Intro to Programming with C - Part 1* course. They are covered in follow-up course: *Intro to Programming with C - Part 2.*

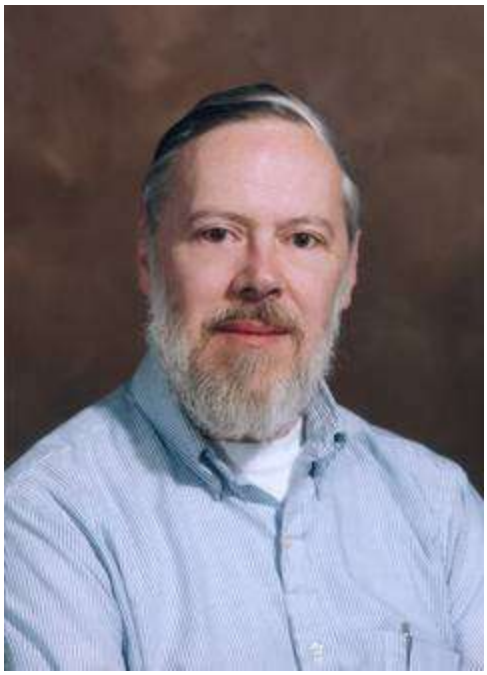
In this course, we will learn how to program in ANSI C. It is a good course for both the novice and experienced programmers. The textbook offers a lot of examples, some good, and some not-so-good. I will try to point out *typos* as much as possible, as this textbook contains quite a few of them. If you notice a typo, please feel free to let me know, in case I had missed it.

One last comment here. I strongly believe in the issue of *readability* of your software. That is, you can write a program that works, and works well, but if it is not *readable,* then you might as well have handed in a program that did not work (almost!).

Readability is just as important as (and I would argue sometimes *more* important than) correctness. If your code is not commented well, if it is not indented well, and if it does not follow other very straight-forward standards for readability (such as descriptive variable names), then your program will be very difficult to read, debug, and modify. This is useless code. 99% of all code that is written is eventually looked at again, by either the original programmer, or by someone else. Code needs to be updated for a variety of reasons (bugs, upgrades, etc.). In just a few short days, you can easily forget why you used a particular segment of code. So, it is so important to follow all of the readability standards when you code. I will stress this all along the way. This is very important to me, because it is very important in industry. If you are going to learn programming from me, I want to be sure that you are learning the right way to program.

This takes a bit more time and effort, but as you will soon see, it is time well spent!

Do you know who this is??



Dennis Ritchie!