0.3 — Introduction to C/C++

Before C++, there was C

The C language was developed in 1972 by Dennis Ritchie at Bell Telephone laboratories, primarily as a systems programming language (a language to write operating systems with). Ritchie's primary goals were to produce a minimalistic language that was easy to compile, allowed efficient access to memory, produced efficient code, and was self-contained (not reliant on other programs). For a high-level language, it was designed to give the programmer a lot of control, while still encouraging platform (hardware and operating system) independence (that is, the code didn't have to be rewritten for each platform).

C ended up being so efficient and flexible that in 1973, Ritchie and Ken Thompson rewrote most of the Unix operating system using C. Many previous operating systems had been written in assembly. Unlike assembly, which produces programs that can only run on specific CPUs, C has excellent portability, allowing Unix to be easily recompiled on many different types of computers and speeding its adoption. C and Unix had their fortunes tied together, and C's popularity was in part tied to the success of Unix as an operating system.

In 1978, Brian Kernighan and Dennis Ritchie published a book called "The C Programming Language". This book, which was commonly known as K&R (after the authors' last names), provided an informal specification for the language and became a de facto standard. When maximum portability was needed, programmers would stick to the recommendations in K&R, because most compilers at the time were implemented to K&R standards.

In 1983, the American National Standards Institute (ANSI) formed a committee to establish a formal standard for C. In 1989 (committees take forever to do anything), they finished, and released the C89 standard, more commonly known as ANSI C. In 1990 the International Organization for Standardization (ISO) adopted ANSI C (with a few minor modifications). This version of C became known as C90. Compilers eventually became ANSI C/C90 compliant, and programs desiring maximum portability were coded to this standard.

In 1999, the ISO committee released a new version of C called C99. C99 adopted many features which had already made their way into compilers as extensions, or had been implemented in C++.

C++

C++ (pronounced "see plus plus") was developed by Bjarne Stroustrup at Bell Labs as an extension to C, starting in 1979. C++ adds many new features to the C language, and is perhaps best thought of as a superset of C, though this is not strictly true (as C99 introduced a few features that do not exist in C++). C++'s claim to fame results primarily from the fact that it is an object-oriented language. As for what an "object" is and how it differs from traditional programming methods, well, we'll cover that in later chapters.

C++ was standardized in 1998 by the ISO committee (this means the ISO standards committee approved a document describing the C++ language, to help ensure all compilers adhere to the same set of standards). A minor update to the language was released in 2003 (called C++03).

Five major updates to the C++ language (C++11, C++14, C++17, C++20, and C++23) have been made since then, each adding additional functionality. C++11 in particular added a huge number of new capabilities, and is widely considered to be the new baseline version of the language. Future upgrades to the language are expected every three or so years.

Each new formal release of the language is called a **language standard** (or **language specification**). Standards are named after the year they are released in. For example, there is no C++15, because there was no new standard in 2015.

C and C++'s philosophy

The underlying design philosophy of C and C++ can be summed up as "trust the programmer" -- which is both wonderful and dangerous. C++ is designed to allow the programmer a high degree of freedom to do what they want. However, this also means the language often won't stop you from doing things that don't make sense, because it will assume you're doing so for some reason it doesn't understand. There are quite a few pitfalls that new programmers are likely to fall into if caught unaware. This is one of the primary reasons why knowing what you shouldn't do in C/C++ is almost as important as knowing what you should do.

Q: What is C++ good at?

C++ excels in situations where high performance and precise control over memory and other resources is needed. Here are a few common types of applications that most likely would be written in C++:

- Video games
- Real-time systems (e.g. for transportation, manufacturing, etc…)
- High-performance financial applications (e.g. high frequency trading)
- Graphical applications and simulations
- Productivity / office applications
- Embedded software
- Audio and video processing
- Artificial intelligence and neural networks

Q: Do I need to know C before I do these tutorials?

Nope! It’s perfectly fine to start with C++, and we’ll teach you everything you need to know (including pitfalls to avoid) along the way.

Once you know C++, it should be pretty easy to learn standard C if you ever have the need. These days, C is mostly used for niche use cases: code that runs on embedded devices, when you need to interact with other languages that can only interface with C, etc… For most other cases, C++ is recommended.

Next lesson
0.4Introduction to C++ development
Back to table of contents
Previous lesson
0.2Introduction to programming languages