The Evolution Of C To C++ Programming

Jazmaine Requeña

Miami Dade College

COP1334

Prof. Stefan Moore

April 3, 2024

Abstract

The evolution of C programming languages shows the advancement of computing technology, reflecting the changing needs and challenges faced by software developers. C++ is a variation of C and is a flexible programming language that has played an important role in software development since its introduction. This research paper will cover an overview of C++ programming, including its evolution, essential features, and many applications. It will include the language's history from its origins in C and how it evolved into current languages. Characteristics that make C++ a popular choice for various applications, including system programming, game development, and embedded systems will be mentioned. In addition, the paper covers C++'s influence on the software industry, including its strengths, problems, and future opportunities.

The Evolution Of C To C++ Programming

As a Computer Information Systems student, I was under the belief that programming languages such as Java, JavaScript, and Python are the norm in programming, and anything else is legacy. However, I was proven wrong after researching the application of C++ in modern society. In terms of test and measurement applications, C, and its variants are the preferred choice of languages, but it can also be used for game development. An example is an open-ended game, Pioneer, which is like a free-roaming app that allows players to explore space. This program is comprised of 58% C++ and 27.7% C, and programmers are welcome to edit and add code to help the functionality of the game. Although it is not widely known to be used, C is ever present and is a foundational language to master other languages.

The C programming language was developed in 1972, by Dennis Ritchie at Bell Laboratories. Originally, it was developed to be used with the Unix operating system and designed to write programs like compilers, operating systems, and text editors (Kernighan, 1983). However, in a few years, it became more than that. It became used in other systems besides Unix and other applications like database systems, telephone switching systems, numerical analysis, etc. It was first developed as BCPL, which wrote compilers for other languages. The creators, Ken Thompson and Dennis Ritchie, wanted a way to easily port the code to other machines without having to rewrite it from scratch. To achieve this, Thompson created B, a simplified version of the BCPL language, designed for recursive, non-numeric, and machine-independent applications like system and language software. B introduced the increment and decrement operators (++ and --). This determined whether the variable change happens before or after noting the value of the operand and is still used today in its evolved languages. B was, however, discontinued due to limited memory management and the lack of data typing. This meant that different types of objects and variables were not specified which could lead to unpredictable results. So, the transition from B to C made a compiler that could create programs quickly and compactly, just like assembly language.

The strengths of C programming were the powerful low-level capability to optimize code, the highly portable and extensive set of code reuse tools, and general-purpose and industry-specific development environments (Logan, 2008). To expand on that, C provides direct access to operating-system or hardware-specific function calls, which allows the user to control specific operations and how many of it is performed at the machine level. The size of the program and its optimization speed can also be controlled due to the optimized compilers. A C

program can also be written to follow other industry standard languages. This allows it to be read by and work on different types of computers and operating systems, without drastic changes needed with the code. Many tools are used to write C code, for example, Microsoft Visual Studio Code and Eclipse. These integrated development environments (IDEs) would often provide features and extensions that would make writing C code, and its variants, much easier to do. C allows direct control of hardware and memory, making it great for applications needing tight space and speed requirements or heavy signal processing. Over 70% of embedded systems use C, although graphical and object-oriented languages also handle common C challenges well.

Despite the strength C brought to the programming community, it has also brought challenges. One drawback is using object-oriented programming. Object-oriented programming allows the creation of new data types and defines actions that can be done with them. So, you would organize your program into pieces called objects. These objects can represent real things, like pieces of equipment. Each object has its properties and behaviors. A scenario Wendy Logan used in her article, *Is C Dead?*, was a test station checking different parts of a circuit board. “You can create a class to represent a component and a method that is part of the class to test the component. You then can create a class to represent the circuit board, which is made up of an array of components. To test your board, you simply call the method responsible for testing each of the components that make up a board.” Although C can use object-oriented programming, it would have to be done manually because it does not have automatic, built-in support. In C, the user has direct control over memory, which is good, however, they would be responsible for freeing memory themselves. Without freeing memory once finished with the program can lead to it becoming slow and less predictable. Other languages would automatically clean the unused

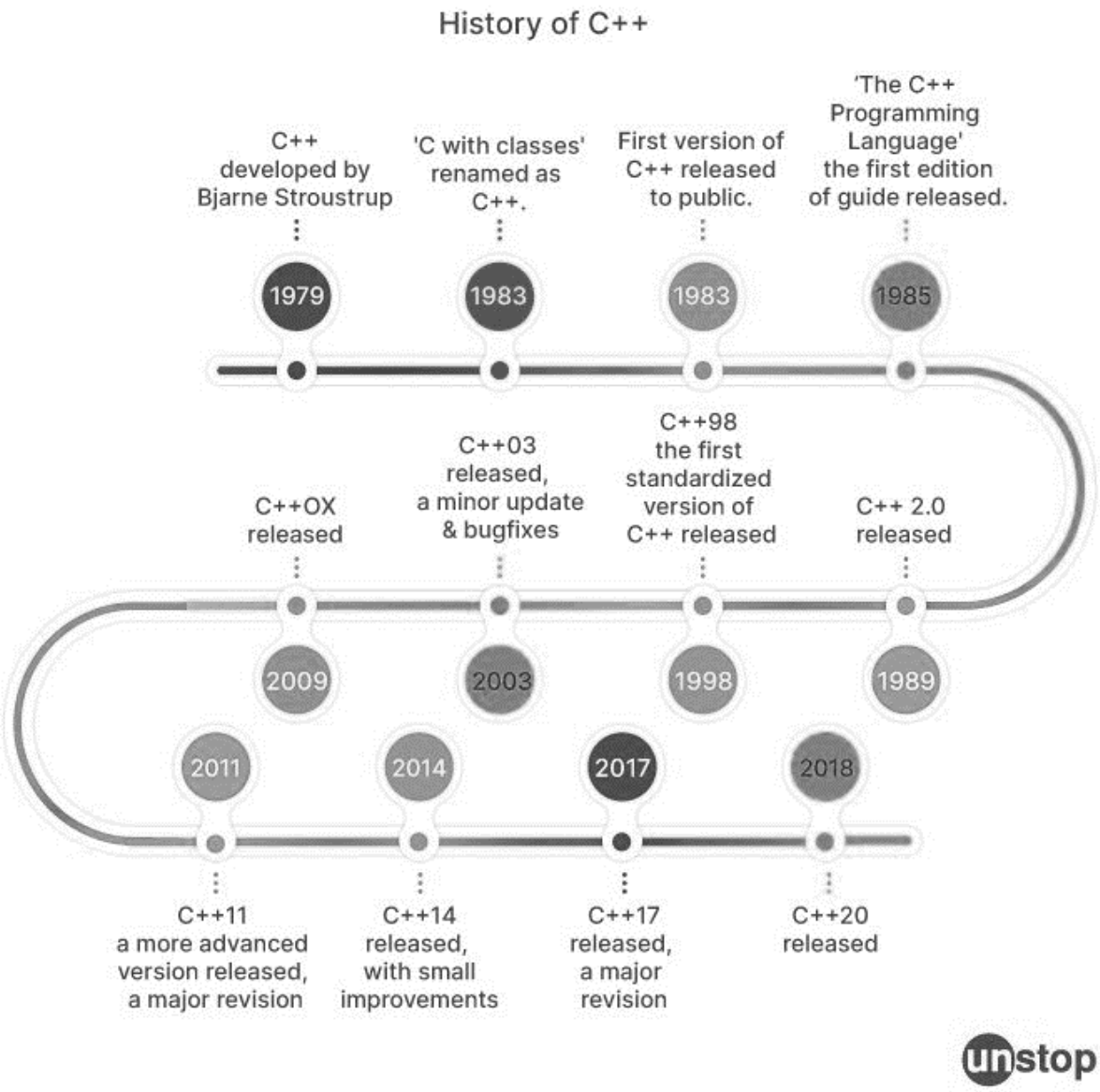
memory once it's needed, but that is not the case for C. Another drawback is its lack of built-in multithreading. As stated in the article, multithreading allows tasks to be performed simultaneously, which enables smoother user interfaces and efficient handling of tasks without slowing down the program. External libraries and extensions can be used to implement multithreading into C; however, this could make the output code more complex and prone to errors.

Despite the drawbacks C may have, it is imperative to remember that C was created to be simple and efficient. Its minimalistic features are what made it so versatile to different systems and gave it room to evolve into different languages based on the needs of the programmer. One of C's predecessors is C++, and it solved some of the problems that C had. C++ is a language that builds on C and makes it easier to do programming, like solving C's lack of object-oriented programming. C was more focused on being simple and fast, and C++ helps us organize our programs more flexibly, especially when dealing with lots of different parts. While C emphasizes simplicity, efficiency, and speed, C++ focuses more on abstraction. C++ does not force object-oriented design, but it does allow for it. As mentioned previously, object-oriented programming can be implemented in C as well but its automation in C++ makes it simpler and less error prone.

To add to the history of C++, it was initially named "C with classes" and was introduced in 1979 by Danish developer Bjarne Stroustrup with the goal in mind for it to be "the better C". By that, it was developed to build complex systems with high-level abstractions while still providing low-level access to the hardware (Goyal 2024). However, until it was released to public use in 1983, Stoustrup upgraded the language to have additional features. These features included inheritance, abstraction, and polymorphism to make it easier to write efficient, modular,

and reusable code. When publicized, it was renamed C++, coined by Rick Mascitti. Figure 1.1 is from UnStop, which displays a brief timeline of the history of C++ to better understand and visualize.

Figure 1.1



In all, programming is constantly evolving to continuously meet modern standards. This evolution started with BCPL, to B, to C, and finally, to C++. However, it continues to evolve. Variants like C#, Unix's C Shell, and D exist and other languages like Java and Python used some of the same concepts of C to be developed into their programming language. It has been possible for C++ to significantly enhance support for conventional C-style programming without introducing more overhead. Furthermore, C++ offers enough language support for object-oriented programming and data abstraction in demanding real-world applications that are high in terms of application complexity and system usage. C++ is still advancing to be more efficient for programmers, but that will be possible due to the trend of developing tools that can make programming code shorter, cleaner, and more maintainable.

Works Cited

Brunner, Tibor, and Zoltán Porkoláb. “Programming Language History: Experiences Based on the Evolution of C++.” *Proceedings of the 10th International Conference on Applied Informatics*, no. 10.14794/ICAI.10.2017.63, Feb. 2017, https://doi.org/10.14794/icai.10.2017.63. Accessed 3 Apr. 2024.

Goyal, Shivani. “The History of C++ (with Timeline Infographic) // Unstop (Formerly Dare2Compete).” *Unstop.com*, 17 Apr. 2023, unstop.com/blog/history-of-cpp. Accessed 3 Apr. 2024.

Logan, Wendy. “Is C Dead?” *EE: Evaluation Engineering*, vol. 47, no. 5, May 2008, pp. 44–49. *EBSCOhost*, search.ebscohost.com/login.aspx?direct=true&AuthType=shib&db=a9h&AN=31929877&site=ehost-live.

Ritchie, Dennis. “Chistory.” *Bell-Labs.com*, Dennis M. Ritchie, 2003, www.bell-labs.com/usr/dmr/www/chist.html. Accessed 3 Apr. 2024.

Stroustrup, Bjarne. “An Overview of C++.” *Proceedings of the 1986 SIGPLAN Workshop on Object-Oriented Programming* -, vol. 21, Oct. 1986, https://doi.org/10.1145/323779.323736. Accessed 3 Apr. 2024.