Overview of C++

C++ can be viewed as updated version of C, adding new features to an already very solid foundation. This includes the low-level memory management, what gives the C and C++ edge in performance and efficiency, compared to some other languages, such as Java or Python, that rely on "garbage collector" subsystem. This feature does exist in C++, but external tools are needed, which are not very efficient, can cause losses in performance, so most developers will stick to manual memory management. This is the very reason why most game engines are built on C++, to allow the developer squeeze out every drop of performance possible. Another way to compare which language is the most effective, is to compare them based on time, memory and energy. Article published by the Universidade do Minho in Portugal, compared most popular programming languages how much energy, time and memory was needed to complete certain tasks. The results were very fascinating, with C being the clear winner, and the C++ right behind him. The main reason for this, was combination of manual memory management and the requirement of strong typed language. This efficiency does not come without a cost. Some developers have noted, learning and mastering C++ can be very difficult and bothersome. Some languages offer the ability to do things with few simple lines of code, but thanks to strong structure here we cannot do things as fast as we want. Every little detail must be described. What type of variable are we dealing with, how long a certain array must be or even how many bytes of data we want to allocate to a certain function. Portability of C++ is strong, meaning that it can run on most system, but it is nowhere close to Java capabilities. Setting up the development environment, for some Windows systems it can be tedious and sometimes nerve-racking. In Linux most of the heavy lifting is already been done natively. Another major is issue is the standard. Are we dealing with C++98, C++17 or C++03. When running the same code, but using a different standard, many things can happen. For example, lambda expressions or automatic type deduction and decltype. In older version of C++, such as C++03, during the declaration we had to specify the type of an object. In C++11 you don't have to do it anymore using "auto" method, which will identify what type of variable are we dealing with. When looking C++ reliability, program verification, orthogonality, data and procedural and syntax and semantics, we can see a how they are all connected to each other. Because of the strict structure and the need of defining every variable with a certain type can be intimidating for those who are just starting to program, but in the long run, it gives a very detailed view how everything is done and can save a lot of time. In python we don't need describe whether our variable is an integer or float. Because of the type safety, the complier will not allow to compile programs, that have not properly typed, which can cause core dumps, ending in a crashed program. Strongly typed language allows us to minimize the orthogonality, meaning that one operation won't affect another. For example. An array can be returned if it is inside a structure. In conclusion, C++ is a more modern version of C, bringing new features, such as object orientated programming, but at the same time retaining the good old solid fountation that the C was built upon. Manual memory management, allowing us to write code for any purpose as needed, very portable. Those features do come at a cost, such a the time needed to master C++, setting up development environment, have a deep understanding of syntax and how thing work on low level, very close to hardware. Despite the negatives, it is still quite popular, maybe not as popular as Java, but still very competive, especially in embedded system, servers and game development.