Language Evaluation: C++

Robert Krency

January 24, 2022

1 Brief History

During his Ph.D. work in the late 1970s, Bjarne Stroustrup was working with the relatively new concept of object-oriented programming. His work at the time heavily used the Simula language, one of the first object-oriented languages. However, Simula proved to be inefficient and slow. Wanting to leverage the speed of the C language and the object-oriented paradigm, Stroustrup began work on a superset of C which he termed C with Classes.

Thus, the first iterations of C++ were born. The name would be changed from C with Classes to C++ in 1983, as a sort of play on words using the increment operator from C. C++ was built to leverage the portability and efficiency of C while adding in features such as classes, inheritance, and strong type checking.

Over the next 15 years, the language would receive many new features supported by a variety of different compilers. In 1998, the first C++ international standard was published by the C++ standards committee as ISO/IEC 14882:1998, and came to be known as C++98. This would also see the inclusion of the Standard Template Library to the C++ standard.

Currently, the standards committee issues a new standard approximately every three years. These standards contain new features and implementations, with a lot of focus on improving the Standard Template Library. C++ remains a widely used language with a rich feature set, and is ranked #4 on the TIOBE rankings.

2 Readability

As a sort of superset of C, C++ uses the popular and easily recognized C-style syntax. For developers familiar with this syntax, it can be easy to get the basic gist of a codebase. C++ provides a lot of leeway to developers to make codebases easy to read, with abstractions such as namespaces. Most keywords are fairly ubiquitous in the programming language world and easy to understand, such as if-statements and for-loops.

However, C++ comes with certain drawbacks that make it difficult to reason about directly. The first thing many developers struggle with is the concept of pointers and references. This leads to a lot of indirection about what a variable explicity represents, and a novice will spend a lot of time deciphering references from values.

Other features such as operator overloading can add a layer of confusion when used with types that are not explicit. Inheritance, polymorphism, and virtual functions add a lot of mental overhead when reasoning about a codebase. The type system of C++ does add guarantees on what type(s) a variable is, helping the reader somewhat.

Overall, the readability of C++ is good. Its syntax is based on a popular style and readability. It provides reasonable abstractions for keeping different software components separate and understandable. Ultimately, the readability comes down to the style and consistency of the

developers.

3 Writability

C++ provides a large amount of features for the developer to utilize. This makes for a very expressive language with useful abstractions, such as namespaces and classes.

Prominent among those are the object-oriented features. This allows for easy reasoning to build data based on real world examples. Many design patterns heavily rely on the presence of inheritance and polymorphism.

Operator overloading allows developers to manipulate data in direct ways with native syntax, meaning a more consistent and less clunky style. For example, building custom arithmetic classes allows the overloading of the '+' operator for addition instead of having to build and call a separate function.

Templates are a mechanism to build more generic data structures and functions, resulting in less duplicate code. This typically results in easier reasoning about a certain data structure's implementation. Less duplicate code also results in less edits and less bugs for future changes.

C++ provides a lot of tools and features to make a developer's life easier. The Standard Template Library provides a lot of performant and standardized tools, such as data structures and algorithms. This library is built into all standards compliant compilers, which results in code that is inherently portable.

Perhaps one of the greatest hurdles to development in C++ is the memory system. C++'s basic types have no guarantees on memory safety. Developers often screw up bounding, or overflow buffers, resulting in undefined and erroneous behavior.

4 Reliability

C++ is typically considered a reliable language and is used in a large variety of performance critical applications.

C++ is a statically typed language, meaning that type checking is done during compile time. This provides guarantees on type operations, and eliminates any runtime errors related to types. This static analysis greatly reduces erroneous development.

One of the great criticisms of and sources of bugs in C++ codebases is memory safety. C++ provides no inherent mechanisms for guarantees on memory safety. Mozilla reports that nearly 70% of its critical security issues are related to memory safety.

The language does however provide mechanisms for error handling. These often come with performance overhead, however. As with most decisions, the trade-offs must be weighed.

5 Implementation Methods

C++ is a compiled language. Any of the standards compliant compilers available thus translate the human written and readable code into machine executable code. This also means that a lot of checks are safety guarantees are done at compile time, which is helpful in entirely eliminating certain categories of bugs.

This results in the tradeoff of fast, efficient code but slow build and compile times. Slow compile times could also result in testing new builds taking long amounts of time. Some language features, such as templates, can also make compiling take drastically longer. Recent work has been done to allow for modularization and faster build times.

One major feature in this area is the concept of header-only libraries, where code for a library gets directly built into the executable every time instead of linked after compilation. This results in efficient code, but longer compile times.

Another benefit of compiled languages is the concept of compile time computation. A lot of mathematics operations may be able to be done at compile time, cutting down on the amount of operations performed at runtime. C++ uses templates to build out specific versions of data structures for each use case, meaning long compile times but performant runtime.

The toolchain for building code into an executable program can often be a source of confusion for novices. However, the speed increases and type guarantees that come with a compiled language make it a worthy tradeoff.