History of C++

References

- https://en.wikipedia.org/wiki/C%2B%2B
- http://www.trytoprogram.com/c-programming/c-programming-history/
- http://www.trytoprogram.com/cplusplus-programming/history/
- https://en.cppreference.com/w/cpp/language/history
- https://en.wikipedia.org/wiki/Simula
- https://dl.acm.org/doi/abs/10.1145/3386320 (written by Bjarne in 2020, history from 2006 to 2020)

Introduction

The history of C++ begins with the C language which was developed by Dennis Ritchie in 1972 while he was working at AT&T's Bell Laboratory. The purpose for creating the C language was because of his work in operating systems, which were still typically implemented in Assembly at the time. C was initially used to implemented the UNIX operating system on PDP mini-computers. The name C comes from the naming of two previous languages, BCPL and B. Given that C comes after B in the English alphabet and that it is the next letter in the BCPL name, the name was born and has stuck ever since.

In 1979 Bjarne Stroustrup began development of a "C with classes" language as part of his PhD work. The idea of adding classes to the language came from a language called Simula, but also providing influence were some other languages: ALGOL, Ada, CLU, and ML (but not all from the start, but over time they have provided influence). By 1983 the language was named C++ and had some additional features added to it like inheritance, polymorphism, and a stronger type system. 1985 saw the first commercial release of C++; the CFront compiler. CFront took C++, translated it to C, then used a C compiler to create an executable.

1989 saw the 2.0 release of the language, along with the CFront compiler. This added multiple inheritance, protected access, abstract classes, and the class specific new/delete operators.

The language continued to slowly evolve until 1998 when the first ISO C++ standard was released, which is known as C++98. By this time the language had a boolean type, exceptions, templates, and namespaces. This release also included the C++ Standard Template Library. In 2003 a small "bug fix" update to the C++98 standard was released, but not much happened with the language until 2011.

In 2011 C++11 standard was ratified. It took several years for the various compiler vendors to get caught up with this standard. This standard represented a huge move forward with the language and signaled the start of an aggressive new interest in evolving and maintaining for the future. The C++11 standard brought many new features, including, inferred data types, generalized constant expressions, lambdas, move operations, range-based for loops, initializer lists, concurrency, smart pointers, regular expressions, and a lot more!

With C++11 being a massive update to the language, the next specification C++14 was a minor update to address some "bugs" in the C++11 standard while also adding a few new language features. Some new features include, variadic templates, function return type deduction, digit separators, and generic lambdas.

In 2017 the C++17 standard was released, again, it is a modest update in preparation for what is expected to be a much larger update to the language in 2020 with the C++20 standard. C++17 brings new things like initializers in if and switch statements, improvements to auto type deduction, nested namespace definitions, a compile-time static if (that can work with

constexpr for compile time decision making), and structured binding declarations that allow for things like returning two values from a function into two inferred typed variables.

Important Note: C++ is not strictly a superset of the C language. For example, C provides for variable length arrays, it also allows the implicit conversion of a `void*` to other types, whereas C++ doesn't. Additionally, C++ reserves language keywords like `new` and `delete`, which can be used as identifiers in C.