**History of the C ++ language**

#### History of creation

The language originated in the early 1980s, when Bell Labs employee Björn Stroustrup came up with a number of enhancements to the C language to suit his own needs. When Stroustrup began working at Bell Labs in the late 1970s on queuing problems (as applied to modeling telephone calls), he found that attempts to use the modeling languages existing at the time were ineffective, and the use of highly efficient machine languages was too difficult because of their limited expressiveness. For example, the Simula language has such features that would be very useful for developing large software, but is too slow, and the BCPL language is fast enough, but too close to low-level languages and is not suitable for developing large software.

Recalling the experience of his dissertation, Stroustrup decided to complement the C language (the successor to BCPL) with the capabilities of the Simula language. C, the base language of the UNIX system on which Bell computers ran, is fast, feature-rich, and portable. Stroustrup added the ability to work with classes and objects to it. As a result, practical modeling problems turned out to be available for solving both in terms of development time (thanks to the use of Simula-like classes) and in terms of computation time (thanks to the speed of C). First of all, C added classes (with encapsulation), class inheritance, strong type checking, inline functions, and default arguments. Early versions of the language, originally called "C with classes", have been available since 1980.

While developing C with classes, Stroustrup wrote cfront, a translator that converts the C source code with classes into plain C source code. This allowed us to work on a new language and use it in practice, using the infrastructure already available in UNIX for C development. The new language, unexpectedly for the author, gained great popularity among colleagues and soon Stroustrup could no longer personally support him, answering thousands of questions.

When creating C ++, Björn Stroustrup wanted

* Get a generic language with static data types, efficiency and portability of the C language.
* Directly and comprehensively support a variety of programming styles, including procedural programming, data abstraction, object-oriented programming, and generic programming.
* Give the programmer freedom of choice, even if it gives him the opportunity to choose the wrong one.
* Maintain compatibility with C as much as possible, thereby making possible an easy transition from programming to C.
* Avoid confusion between C and C ++: any construct that is allowed in both languages must mean the same in each of them and lead to the same program behavior.
* Avoid features that are platform dependent or not universal.
* Don't pay for what's not used — no language tool should degrade the performance of programs that don't use it. Don't require an overly complex programming environment. The choice of C as the basis for creating a new programming language is explained by the fact that the C language:

1.is a multipurpose, concise and relatively low-level language;  
2. suitable for most system tasks;  
3. performed everywhere and on everything;  
4. docks with UNIX programming environment.

- B. Stroustrup. C ++ programming language. Section 1.6

Despite a number of known shortcomings of the C language, Stroustrup decided to use it as a basis, since "C has its own problems, but a language developed from scratch would have them, and we know C problems." It also made it possible to quickly develop a prototype compiler (cfront) that only translated the added syntax elements into the original C language. As C ++ was developed, other features were included that overlapped the capabilities of C constructs, and therefore the question of abandoning language compatibility by removing obsolete constructs. However, compatibility has been maintained for the following reasons:

* preservation of the current code, originally written in C and directly ported to C ++;
* elimination of the need to retrain programmers who previously studied C (they only need to learn new C ++ tools);
* elimination of confusion between languages when they are used together ("if two languages are used together, their differences should be either minimal, or so large that the languages cannot be confused").

By 1983, new features were added to the language, such as virtual functions, function and operator overloading, references, constants, user control over free memory management, improved type checking, and a new style of comments (//). The resulting language has ceased to be just an extended version of classic C and has been renamed from C with classes to "C ++". Its first commercial release took place in October 1985.

The resulting language name comes from the C ++ unary postfix increment operator (increasing the value of a variable by one).

Prior to official standardization, the language was developed primarily by Stroustrup in response to requests from the programming community. The function of standard language descriptions was performed by the published works on C ++ written by Stroustrup (language description, reference manual, and so on).

#### History of standards

In 1985, the first edition of The C ++ Programming Language was released, providing the first description of this language, which was extremely important due to the lack of an official standard.

In 1989, C ++ version 2.0 was released. Its new features included multiple inheritance, abstract classes, static member functions, constant functions, and protected members. In 1990, the C ++ Commented Reference Guide was published, which later became the basis of the standard. Recent updates have included templates, exceptions, namespaces, new casts, and booleans.

The C ++ Standard Library has also evolved with it. The first addition to the C ++ standard library was I / O streams, which provide a means to replace the traditional C printf and scanf functions. Later, the most significant development of the standard library was the inclusion of the Standard Template Library.

In 1998, the ISO / IEC 14882: 1998 language standard (known as C ++ 98) was published, developed by the C ++ standardization committee (ISO / IEC JTC1 / SC22 / WG21 working group). The C ++ standard does not cover naming conventions for objects, some of the details of exception handling, and other features related to implementation details, which makes the object code generated by different compilers incompatible. However, to do this, third parties have created many standards for specific architectures and operating systems.

In 2003, the ISO / IEC 14882: 2003 language standard was published, where the identified errors and shortcomings of the previous version of the standard were corrected.

In 2005, the Library Technical Report 1 (referred to as TR1 for short) was released. While not officially part of the standard, the report describes extensions to the standard library that the authors expected to be included in the next version of the C ++ language. TR1 support is being improved in almost all supported C ++ compilers.

Since 2009, work has been carried out to update the previous standard, the preliminary version of the new standard was first C ++ 09, and a year later C ++ 0x, today - C ++ 11, which included additions to the core of the language and an extension of the standard library, including most of TR1.

C ++ continues to evolve to meet modern requirements. One of the groups developing the C ++ language and sending suggestions for improving it to the C ++ Standardization Committee is Boost, which is engaged in, among other things, improving the capabilities of the language by adding metaprogramming features to it.

Nobody owns the rights to the C ++ language, it is free. However, the language standard document itself (with the exception of drafts) is not available free of charge.

Source: <http://cpp-cpp.blogspot.com/2014/01/c.html>

overloading – перегрузка в языках программирования высокого уровня (например, в С++) - использование одного и того же идентификатора для обозначения различных операций, процедур или методов. Транслятор выбирает необходимую процедуру на основании числа и типов параметров.