# The Evolution of C++: A Comparative Study with C and its Futuristic Development

Yinhao Fang
231502010@smail.nju.edu.cn
Nanjing University
Nanjing, Jiangsu, People's Republic of China

## Abstract

After a comprehensive analysis of the origin and development of C++, it is indisputable that its potential cannot be ignored. This paper explores the historical progression of C++, its core differences from C, and key enhancements introduced in the updates C++98 and C++11. Furthermore, the philosophical duality of Object-Oriented Programming (OOP) and Procedural Programming paradigms are contrasted, focusing particularly on C++. Finally, the future development trends of C++ are evaluated amidst the increasing popularity of C and Java.

***Keywords:*** C++, C++98, C++11, OOP

**ACM Reference Format:**
Yinhao Fang. 2023. **The Evolution of C++: A Comparative Study with C and its Futuristic Development**. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 Introduction

C++ and C, although branches of a similar lineage, have diverged profoundly in their stylistic and syntactical elements. The evolution of C++ includes significant shifts, primarily driven by the transition from procedural to OOP.

This study evaluates the semantic differences of code written in C++ as compared with C and sheds light on the transition and advancements in programming paradigms. It also aims to explore the trend of C++'s future development.

## 2 Distinctions between C and C++

Although C++ can be seen as a superset of C, which even abandon some tendency to keep compatible with C, the encapsulation of data and functions into objects in C++ sets it apart from C significantly, elucidating the nuances between the procedural and object-oriented extrapolation of real-world problems.

After its formal separation from C, C++ proved to be an extensible, powerful, and flexible language with significant emphasis on data abstraction, encapsulation, and inheritance.

All I mention below is main differences beginners feel most frequently:

First, class, which is a burden for beginners. The encapsulation of the class makes the beginner tired of the program, feel uncomfortable and troublesome.

Second, references. references in C++ is best to try not to use it, unless absolutely necessary. References are even more confusing for beginners, not knowing which is a reference and which is a variable.

Third, function overload, beginners to learn the function overload seems to be no harm, but this will make the beginners subconscious on the importance of the C language variable type to dilute, to remember that the C language is the most sensitive to the variable type, the importance of the variable type in the C language is self-evident.

## 3 Innovations from C++98 to C++11

1. Initialize the form

(1) C++98 supports: assignment symbol initialization, parenthesis initialization, implicit initialization of custom type entities

(2) New in C++11: Brace initialization

I. When initialized with curly braces, the assignment symbol is optional.

ii, when using curly braces for initialization, some unintended implicit conversions can be disallowed (for example, disallowing truncation from floating-point numbers to integers).

2. Constant

(1) C++98 support: using const to modify the type of an entity is the programmer's promise to never write to the entity.

(2) New in C++11: The use of constexpr to modify the type of the entity is the programmer's suggestion to "calculate the value of the entity at compile time", which helps the compiler to take some optimization strategies for "reading the value of the entity in the process."


*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

I. Expressions evaluated at the compilation stage are collectively referred to as constant expressions. Most commonly used: length when declaring arrays, case branches, some template arguments, initializing constexpr entities.

ii, constexpr can modify the return type of a function. However, the function call expression can only be treated as a constant expression if the function is called with constant arguments. If called with variable arguments, the function call expression cannot be treated as a constant expression.

3. Null pointer

C++98 supports: 0, NULL

(2) New in C++11: nullptr.

i. All pointer types share the same nullptr.

Unlike 0 or NULL, nullptr cannot be implicitly converted to integer type.

etc.

## 4 Understanding 'Object-Oriented Programming'

Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of "objects," which can encapsulate data and behavior. C++, being an extension of the C programming language, introduces powerful features for object-oriented programming.

**Class and Object:** At the core of C++ OOP is the concept of a "class." A class is a blueprint for creating objects, defining their properties (data members) and behaviors (member functions/methods). An "object" is an instance of a class.

```cpp
// Example of a class definition
class Car {
private:
    string brand;
    int year;

public:
    void setBrand(string b) {
        brand = b;
    }

    void setYear(int y) {
        year = y;
    }
};
```

**Encapsulation:**C++ supports encapsulation, which means bundling the data (attributes) and methods (functions) that operate on the data within a single unit, i.e., the class. Access specifiers (public, private, protected) control the visibility of members.

```cpp
class Circle {
private:
    double radius;

public:
    void setRadius(double r) {
        if (r > 0) {
            radius = r;
        }
    }

    double calculateArea() {
        return 3.14 * radius * radius;
    }
};
```

**Inheritance:**Inheritance allows a class to inherit the properties and behaviors of another class. This promotes code reusability and establishes an "is-a" relationship between classes.

```cpp
// Base class
class Animal {
public:
    void eat() {
        cout << "Animal is eating." << endl;
    }
};

// Derived class
class Dog : public Animal {
public:
    void bark() {
        cout << "Dog is barking." << endl;
    }
};
```

In summary, C++'s Object-Oriented Programming provides a robust framework for creating modular, reusable, and maintainable code through the use of classes, encapsulation, inheritance, polymorphism, abstraction, constructors, and templates. These features collectively contribute to the flexibility, scalability, and efficiency of C++ programs.

## 5 Future Trends for C++

Note that C++ passing by reference has no special identifier for function calls, which is indeed a language design error. The fundamental reason is that the address operator was reused when the reference syntax was first developed. This flaw is one of the early design flaws of C++, and C++ 's insistence on not adding keywords is actually one of the reasons why it has become so complex, and another reason is to try to be compatible with C.You can see that any language has historical baggage.

C++'s future in the face of growing popularity of C and Java is a point of critical enquiry given the paradigmatic differences these languages propagate and the challenges they independently address. However, the robustness and extensibility of C++ cannot be understated, making its disappearance highly improbable but rather evolution into a more sophisticated toolset.