# 从 C++98 到 C++14：
# 试论 C++ 版本革新对算法竞赛选手的影响

何彦锋

2024 年 12 月 10 日

**摘要**

近二十年来，C++ 语言迅猛发展，引入了许许多多的新语法、新特性，实现了从 C++98 到 C++14 的迅速迭代。对于使用 C++ 进行算法竞赛的选手来说，如此巨大的迭代必将带给他们不可估量的影响。因此，本文将以几个从 C++98 到 C++14 的主要变化为例，探讨 C++ 版本革新提升选手代码编写速度、利于选手写出正确性更好的代码、提升选手代码运行效率等三方面积极影响，希望能推动 C++ 语言的进一步发展，助力 C++ 语言在算法竞赛中的良好应用。

## 1 引言

近二十年间，C++ [8, 7] 语言迅猛发展，不断引入新语法与新特性，实现了从 C++98 到 C++14 的多次重大迭代，对使用 C++ 进行算法竞赛的选手们产生了深远的影响。

在此背景下，已经有不少算法竞赛的选手在他们的博客中梳理了从 C++98 到 C++14 变化，整理出了一些适合在算法竞赛中使用的新特性。例如算法竞赛网站“洛谷”上的这篇文章 [5] 举例介绍了许多 C++14 的新特性，而同在该网站的另一篇文章 [1] 更进一步，甚至梳理到了 C++20 的“语法糖”。然而，这些文章大多仅停留在对新特性的介绍，而很少结合算法竞赛的特殊性，谈论 C++ 版本革新对选手的意义。

因此，本文将首先在第 2 节指出算法竞赛相较于常规开发环境的特殊性，然后在第 3 节以几个从 C++98 到 C++14 的主要变化为例，从提升选手代码编写速度、利于选手写出正确性更好的代码、提升选手代码运行效

率等三个方面，深入剖析 C++ 版本革新对算法竞赛选手的积极作用。我们希望通过本文的研究，能够推动 C++ 语言的进一步发展，助力 C++ 语言在算法竞赛中的良好应用，为算法竞赛选手提供更多有力的支持。

## 2 算法竞赛相较于常规开发环境的特殊性

相比于常规的开发环境，算法竞赛尤其讲究选手的代码编写速度，看重选手代码的正确性，强调选手代码的运行效率。

首先，相较于常规开发环境，算法竞赛更加讲究选手的代码编写速度。在常规开发环境中，程序员们往往有宽裕的时间来慢慢编写自己的代码。然而在算法竞赛赛场上，情况截然不同。由于时间紧迫，选手们越是在代码编写上浪费时间，用于他们思考推理题目的时间就越少，他们想出高质量解答并通过题目的机会就越微小。因此，算法竞赛对选手代码的编写速度有更高要求。

其次，相较于常规开发环境，算法竞赛更看重选手代码的正确性。虽然在常规开发环境中，程序员们也追求代码的正确性，但当程序出现错误时，程序员们往往可以接管程序，用人工的方式确保程序正确运行。而在算法竞赛赛场上，选手一旦提交了自己的代码，就不能再对提交的代码做出任何的改动，即使程序运行有错，选手也不可能人工接管程序，确保程序正确运行。正因如此，算法竞赛尤其强调选手代码在面对各种数据时都能保持正确性。

最后，相较于常规开发环境，算法竞赛更凸显代码运行效率的重要性。在常规开发环境中，程序员们往往不需要追求极致的运行效率。但是在算法竞赛中，程序的运行效率尤为重要，有时哪怕是在常数上对算法时间复杂度的微小优化，都是决定竞赛胜负的关键因素。因此，算法竞赛会更注重代码的运行效率。

## 3 C++ 版本革新对算法竞赛选手的意义

相比于 C++98，C++14 新增了许多内容。接下来，本文将从三个方面论述 C++ 版本革新对算法竞赛选手的积极影响，并举若干实例予以论证。

### 3.1 提升选手的代码编写速度

由于算法竞赛对选手们代码编写速度的高要求，选手们往往采取减少代码量等方式提升自己的代码编写速度。以在 C++11 及以后的 C++ 版本中被支持的类型推断关键字 `auto` [4] 为例。`auto` 是一种静态类型推断关键字，它可以在对象的初始化过程中推断出对象的静态类型。在算法竞赛的场景中，它可以代替许多极为复杂的类型名，减少选手的代码量，从而提升选手的代码编写速度。

如下的两段代码都实现了输出集合 `set<int> st` 当中元素的功能，但前一段代码使用 C++98 的语法编写，后一段代码使用了关键字 `auto`，属于 C++11 及以后版本的语法：

```
1    //C++98
2    for(set<int>::iterator iter = st.begin();iter != st.end();++iter) cout<<*iter<<" ";
```

```
1    //C++11 and later version using auto
2    for(auto iter = st.begin();iter != st.end();++iter) cout<<*iter<<" ";
```

如果使用 `auto` 关键字搭配 C++11 及以后版本支持的 **range-for** 语法，那么代码量还可以进一步缩减，选手写出相同功能的代码的速度还能进一步提升。

```
1    //C++11 and later version using auto and range-for
2    for(auto num : st) cout<<num<<" ";
```

### 3.2 有利选手写出正确性更好的代码

在算法竞赛中，被选手们提交的代码会在与外界隔离的平台运行，无法受到选手操控。因此，选手们只有写出正确性足够好的代码，才能保证代码在运行时不出错。

当选手不得不使用基于伪随机数的算法来尝试通过某道题目时，代码的正确性就显得尤为关键，因为只有当伪随机数函数生成的数具有良好的随机性时，这份代码才有可能通过该题目。然而，在 C++98 版本里，由于 C++ 标准未对伪随机数函数 `rand()` 生成的随机数的质量提出要求，所以在不同的机器下，`rand()` 生成的伪随机数可能并不均匀随机，且其可能具有较短的循环周期。同时，在不同的系统下，生成的伪随机数的值域也有所不同，其中 Windows 系统下的值域是 $[0, 2^{15})$，而在 Linux 系统下的值域是 $[0, 2^{31})$。这些问题都会降低选手代码的正确性。

幸运的是，在 C++11 及以后的版本中，C++ 提供了一个专门用于生成伪随机数的头文件 `random`。在这个头文件下，C++ 定义了一个基于梅森缠绕器的随机数类 `mt19937`，调用这个随机数类不但能产生质量更高的伪随机数，而且伪随机数会具有更广的值域，达到了 $[0, 2^{32})$ [6]。同在该头文件下还有另一个类似的随机数类 `mt19937_64`，它生成的伪随机数质量和前者类似，但伪随机数的值域扩大到 $[0, 2^{64})$。有了这样高质量的为随机数函数的加持，选手将能写出正确性更好的代码，从而降低代码运行出错的概率。

### 3.3 提升选手代码运行效率

算法竞赛中不乏一些对程序运行时间效率要求很高的题目。在应对这些题目的时候，不少选手会绞尽脑汁让自己的程序运行得更快。

提升代码运行效率的办法有许多，包括循环展开、编译期优化等方法。这里以编译期优化关键字 `constexpr` 为例。`constexpr` 是一个在 C++11 中引入、在 C++14 中完善的关键字，它允许选手们声明在编译期即可确定其值的变量和函数。通过这一关键字，选手们可以把一部分计算放到编译期完成，从而减少运行时的时间开销。

在算法竞赛网站“洛谷”的两份代码提交记录 [3, 2] 中，第一份代码 [3] 没有使用 `constexpr`，而第二份代码 [2] 在程序开头的几个函数中使用了 `constexpr`。可以观察到，第一份代码用了总计 1.22s 的时间通过所有测试点，而第二份代码仅用了 1.02s 就通过了所有测试点，可见 `constexpr` 在整体上减少了约 16% 的运行时间。同时，在 9 个测试点中，两份代码都在第 9 个测试点花费了最长的时间，前者花费了 467ms 而后者花费了 394ms，可见，在耗费时间最大的测试点上 `constexpr` 也减少了约 16% 的运行时间。

## 4 总结及未来展望

C++ 版本的持续革新为算法竞赛选手带来了显著的积极影响。首先，新关键字如 `auto` 的引入和语法优化提升了代码编写速度，使选手能够更快速地实现算法逻辑，从而在竞赛中抢占先机。其次，C++ 通过革新引入 `random` 头文件等，为选手提供了更高质量的伪随机数函数，有助于选手写出正确性更好的模板代码。最后，通过引入 constexpr 编译期优化关键字

等方法，革新后的 C++ 显著提升了代码的运行效率，使选手能够在严格的时间限制内解决更复杂的问题。综上所述，C++ 版本的革新为算法竞赛选手提供了强大的工具和支持，助力他们在竞赛中取得更好的成绩。未来，随着 C++ 语言的不断演进，我们有理由相信，它将为算法竞赛领域带来更多的惊喜和突破。

然而，需要指出的是，本文仅仅讨论了 C++ 版本革新对算法竞赛选手的积极意义，而没有思考其消极意义。正所谓“凡事都有两面”，C++ 版本革新带来的过多特性会不会增大算法竞赛初学者的学习负担？冷门生僻的 C++ 语法会不会造成代码可读性的降低，增大算法竞赛选手的代码维护成本？这些都是值得讨论且需要讨论的问题。在未来，我希望能在一些算法竞赛选手的社群中展开调查研究，对 C++ 版本革新的负面意义作出一些思考。

# 参考文献


[1] Acc_Robin. 从 C++98 到 C++20，寻觅甜甜的语法糖们. https://www.luogu.com.cn/article/l86yn4aw, 2024.

[2] Computer1828. 使用了 constexpr 的提交记录. https://www.luogu.com.cn/record/189048818, 2024.

[3] Computer1828. 未使用 constexpr 的提交记录. https://www.luogu.com.cn/record/189049402, 2024.

[4] C++ Developers. Placeholder type specifiers (since C++11) - cppreference.com — en.cppreference.com. https://en.cppreference.com/w/cpp/language/auto.

[5] Fidel. 从 C++98 到 C++14 —— OIer 升级指南. https://www.luogu.com/article/7ofveldz, 2021.

[6] gsczl71. rand VS mt19937. https://www.cnblogs.com/gsczl71/p/18132237, 2024.

[7] Bjarne Stroustrup. Evolving a language in and for the real world: C++ 1991-2006. In *Proceedings of the Third ACM SIGPLAN Conference*

*on History of Programming Languages*, HOPL III, page 4–1–4–59, New York, NY, USA, 2007. Association for Computing Machinery.

[8] Bjarne Stroustrup. Thriving in a crowded and changing world: C++ 2006–2020. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020.