
简介 Intro

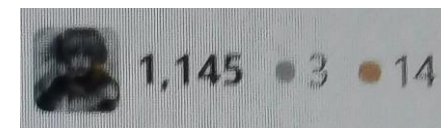
现代C++基础 Modern C++ Basics

Jiaming Liang, undergraduate from Peking University

Preface – Who am I?

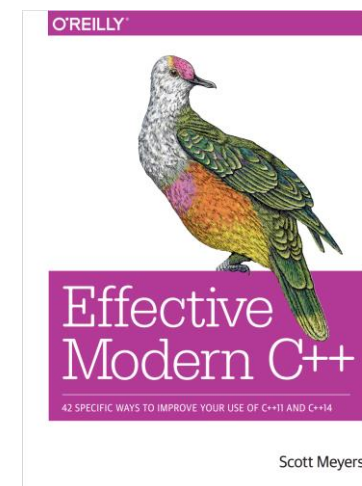
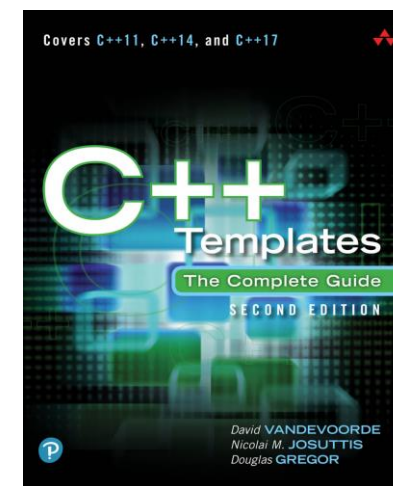
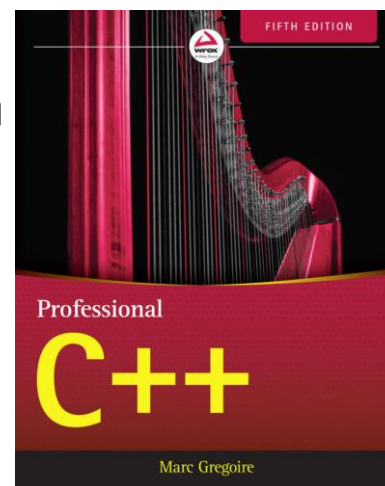
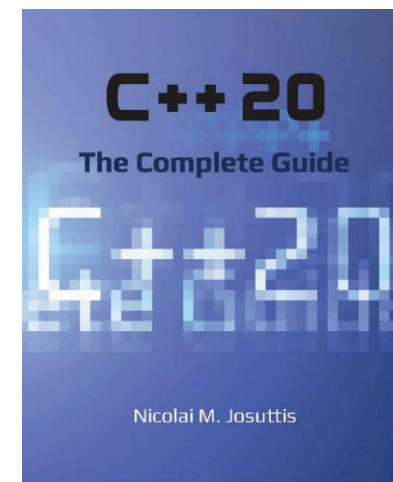
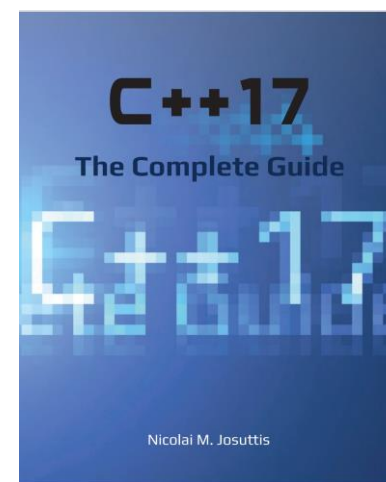
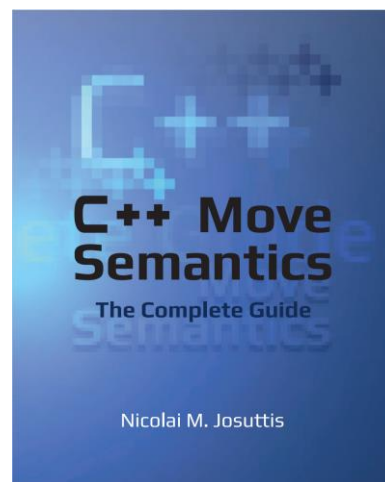
- 20岁，是学生
 - 北京大学-信息科学技术学院-2020级本科生
 - 北京大学奖学金
 - 保研至计算机学院
 - 免试硕士研究生第一名
 - etc.
- 荣誉可能有很多，但是这些无关紧要...
 - 既然这门课是C++相关的，C++上的或许更重要一些

Preface – Why can I?



@2023.10

- 我完整读过大量C++相关书籍和技术文章
 - 这些书基本都在600-1000页左右
 - 如果你是个正常本科生的话，基本不会花费这么多时间读一门语言的材料。——(当然我不正常)——
- 我密切关注CppCon和最新C++标准，参与大量社区讨论
 - stackoverflow ~1.2K reputation
 - 知乎应该算是熟面孔了吧（大概）
- 编写过相对有一定体量的project
 - 总之就是读/写过大量C++代码...



Preface – Why do I?

- 所以我为什么开这门课程呢？
 - 说实话，在公共平台发布课程，我实在诚惶诚恐，担心自己水平有限。
 - 虽然相对其他本科生，我大概还是有一定资格的；但是我没有经历过超大型项目的训练，因此在经验上确实不足，恐怕课程中有大量可改进的地方（这也是为什么课程称为“基础”）。
 - 但是，PKU的同学们、乃至国内大部分高校的本科生们，都仅仅学习过上古C++（即C++98），连C++11也是浅尝辄止，我确实感到这实在是一件憾事。
 - 这让大部分同学对C++有极深的误解，认为这是一门垃圾的语言。
 - 或许它确实在某些地方是垃圾的，但是这些地方基本上要深刻地学习了现代C++才能知道。
 - For example:

```
#include <iostream>

int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

What you learn
Ugly and inefficient

```
import std;

int main()
{
    std::println("Hello, world!");
    return 0;
}
```

What C++ code will be like in 2025
(Visual Studio already supports it)

Preface – Who should learn?

- 那么什么样的人比较适合学习这门课程呢？
 - 计算机系做比较底层的人（例如体系结构、操作系统、编译、计算机网络、图形学、高性能计算...大部分是比较底层的，一些软件上的研究可能不需要）、以及电子学院准备搞嵌入式系统的可能也算。
 - 而对于智能系，考虑到大部分人都在做比较顶层的设计，比如神经网络的架构等，而不必考虑底层算子的速度改进，一般不需要学习这门课程。
 - 如果你是做像PyTorch这样的系统的，研究生大概不是在智能学院，而是在计算机学院...
 - 如果你已经参与工作，需要使用C++，而你了解现代C++知识、或者了解的很有限，或许也可以听一听这门课。
 - 当然如果你感兴趣，并且也满足我们马上会讲的先修要求，也欢迎你来上这门课程！
 - 这不是只给所谓“高材生”听的课程；虽然课程会比较难，但是慢慢听、多写代码，总是可以搞明白的。
 - 很多东西也不用完全记住，需要的时候在手册上具体查下就行。

Preface – Why do I?

- 总之，我希望通过这门课程来让大家从学习过的上古C++出发，认识今天的C++是什么样的。
 - 或者说我希望学了这门课，大家就不要再读我读过的那些书，或者速读即可。
 - 课程会涵盖大部分C++11 - C++23的知识，同时简单提及已经确定的、或者十分有希望的C++26特性。
 - 除了下一节的复习课，如果一个特性是C++11/14的，那么我不会把他当成语言的新特性；C++17 - C++23的会有注明。
 - 目前的项目仅能用到C++17的还是比较多的，所以我从C++17开始注明。
 - 如果你还有比较长的时间才会工作，那么你基本不用在意版本，在项目里直接全部用最新的。
- 希望大家在将来可以进入C++标准委员会，不论是作为国人还是程序员...
 - 拥抱进步，发出你们的声音，为gcc/clang等开源社区做出自己的贡献！

Preface

- 这门课有些地方会有很多PPT页...
 - 因为我希望把知识点都放上去，防止大家溜号了就知道了。
 - 同时动画、文字什么的是要在PDF上无重叠的。
 - 虽然如果只把提纲放上去，我一直在讲，可能显得我很牛，但是这不利于大家学习；
 - 我会在讲到合适的时候切换到IDE，写一些代码给大家看用法。

Preface

- 特别声明：
 - 本课程允许转载，只需要注明该视频的出处和我的名字（与私有制决裂...）。准确说，课程使用CC-BY-NC-SA 4.0协议。
- 特别鸣谢：
 - 谷雨，PKU graduate @ 2019级，对本课程PPT的部分审阅和建议。
 - 当然，由于时间原因，谷雨学长未作完全的校对，如果PPT中出现错误，也是我自己的问题。
- 特别教训：
 - 大家不要在Linux office编辑Microsoft的PPT...
 - 虽然我就是加个图...
 - 这导致本节PPT完全重做，因为格式、字体完全木大，内容丢失...总之整个PPT没法用了。

- Requirements
- Why C++?
- Course Outline
- Useful Tools
- Concepts need to be known
- Code style suggestions

Introduction

Requirements

Requirements

- 本课程要求首先具有以下基本知识：
 - 关于C++的基本知识；PKU学生就是学过计算概论（A）、学过程设或者软设这两门课之一。
 - 非PKU学生可以看看以下内容是否学过：
 - 整型、浮点型、数组、结构体、函数、C风格字符串（`char*`）、指针、枚举、运算符；C风格转型（如`(int)1.0f`；不要求掌握C++风格的转型）；基本的循环与分支语句。
 - 这是计算概论（A）的内容；计算概论（B）应该也讲了。
 - 基本的OOP，类相关的。
 - 可访问性，即`private`, `public`, `protected`
 - 构造函数、析构函数的基本用法
 - 多态相关的，包括`virtual`和函数覆写。
 - `this`指针。
 - 继承相关，基类和派生类
 - 基本的IO，包括`cin`, `cout`（其他的如文件操作、`iomanip`不要求）。

Requirements

- 函数重载，包括运算符重载（特别是赋值运算符重载）。
 - 注意，`A a1, a2 = a1;`，这里`a2`调用的是拷贝构造函数；
 - 而之后再使用`a2 = a1`，由于对象已经完整构造，这里的`a2`调用的是拷贝赋值函数。
- 基本的模板知识和STL知识（主要是容器）。
 - 当然STL是一个legacy name。
- 引用（这里指左值引用，如果你知道其他引用方式，我们不要求掌握）。
- 以上是程设/软设都学过的知识。
- 关于计算机系统的基本知识；PKU学生等价于学习过ICS，也等价于学过CS:APP。
 - 平台强相关的不要掌握，例如gcc如何链接、ECF、网络相关（知道就行了）。
- 关于数据结构与算法的基本知识；PKU学生等价于学过数算。
 - THU学生如果学过数据结构这门课也一样。十分向大家推荐邓俊辉老师的数据结构课程！
 - 我们对算法和数据结构的要求不会太高，基本停留在理论复杂度分析上。
- 这些课是PKU大二下之前全部必修的。

Requirements

- 我们在下一节课会进行全面的复习，并进行一些小而杂的扩展。
 - 特别地，课程中注明Optional或者标题上打星号（*）的都是比较难、但是它的难度不匹配重要程度的；一句话：不会也行。
 - 一些具体的API重载记个大概就行，到时候可以看IDE的提示。
- 此外，由于大家以后必然要总是用英语读很多重要的C++材料（如cppreference） ...
 - Thus, this course will use English text.
 - Sentences and words won't be very complicated.
 - CET4 is enough!
 - BTW, never mind my possible syntax error...I'm not a native speaker too.
 - It's totally OK if you aren't good at English, since I'll implicitly translate PPTs during teaching.

Introduction

Why C++?

Why C++?

- It's definitely not enough to only learn C++...
 - Script languages, like Python, Lua, Ruby, etc.
 - Internet-related, like JS, TS, Go, etc.
 - C#, Java...
 - Function programming, like Haskell/Lisp.
 - Data science, like Julia/Matlab, etc.
- Remember, using C++ represents that you choose a powerful and efficient language.
 - E.g. if you do machine learning and only care about network structure, why not Python?
- Let's see how Bjarne Stroustrup, C++ creator, views C++!
 - Credit: InfoQ @ bilibili

Why C++?

- One of the most important philosophy in C++ is zero-overhead abstract (零额外开销抽象).
 - 1. You don't pay for what you don't use.
 - 2. What you do use is just as efficient as what you could **reasonably** write by hand.
- It cannot be denied that code can be optimized compared to standard libraries in a specific platform or occasion.
- But your first choice should always be the standard library, and when the profiler (性能探测器) tells you that it's necessary to do optimization somewhere, do it then.
- Only a few features disobey such philosophy, and we may mention it in the future.

Introduction

Course Outline

Course Outline

- Totally 16 lectures; each lecture is expected to be 150min.
 - Just like a course with 3 credits!
- Introduction (This lecture)
- Basic review & extension
- Containers
- Ranges and algorithms
 - These three lectures may be longer, because you've learnt major parts.
 - We'll go deeper beyond what you've learnt, like learning from source code of some STL implementation.

Course Outline

- Lifetime
 - We'll teach you C++-style type conversion, variant and any, tell right from wrong (especially UB).
- Programming in multiple files
 - We'll teach you header files, source files, linkage, and C++20 modules.
 - And a strong build tool developed by Chinese, Xmake!
- Error Handling
 - We'll teach you optional/expected, exceptions, and how to do unit test.
- String and Stream
 - We'll teach you Unicode, string/string_view, format/print (to substitute rubbish iomanip, `std::cout` or `printf`), stream and regular expression.
- Move Semantics, for 2.5 lectures.

Course Outline

- Templates, for 1.5 lectures.
 - In Move Semantics, we'll teach you return value optimization, move ctor/assignment, rvalue reference, value categories, etc.
 - In Templates, we'll teach you concept, template specialization, variadic templates, metaprogramming, etc.
 - We'll then go back to Move Semantics to see perfect forwarding, etc.
- Multithreading, for 1.5 lectures.
 - We'll teach you threads, semaphores, mutex, locks, condition variable, future-promise model, atomic variables, memory order, etc..
- Coroutines, for 0.5 lectures.
- Memory management
 - We'll teach you advanced techniques like allocators, pmr.
- Final
 - Summary, uncovered features and future vision of C++.

Course Outline

- If you feel unable to learn the whole lecture at once, you may learn sections one by one.
 - When you feel that you have grasped all things, then learn the next section.
- There will be some code examples and homework.
 - 《实践论》：认识的两次飞跃
 - 课程是从实践到认识的第一次飞跃。
 - 作业是从认识到实践的第二次飞跃！
 - 作业必须要做，否则讲的东西下周肯定忘了。

Introduction

Useful tools

Useful tools

- First and foremost, I do NOT encourage illegal actions like visiting websites by VPN.
 - There are various mirror websites.
- [GitHub](#): a code base where you can record & rewind & etc. to cooperate with others, or view & download others' public repos.
 - It's possible that you can visit this website directly, but quite unstable...
 - You can also use [this mirror website](#) (only for IP in China mainland). All `https://github.com/AA/BB` can be accessed by `https://hub.nuaa.cf/AA/BB`.
 - I will also upload code in [GitLab](#), which can be directly accessed.
- [StackOverflow](#): an international forum for programmers.
 - It's also possible to visit using domestic IP but more complex.

Useful tools

- [Cpp reference](#): a semi-authoritative C++ standard document, just like a dictionary in C++; follow up the newest standard.
 - However, it's obscure to some extent, especially for the beginner.
 - It has [Chinese version](#), but quite similar to machine translation...
 - Its search column is duckduckgo (inaccessible in Chinese mainland), you can use Baidu:



Useful tools

- Special notice: you can designate language version, otherwise it may be way too long...

PageDiscussionStandard revision: DiffViewEditHistory

C++Strings librarystd::basic_string

What the hell!?

std::basic_string<CharT,Traits,Allocator>::insert

```
basic_string& insert( size_type index, size_type count, CharT ch ); (1) (until C++20)
constexpr basic_string& insert( size_type index, size_type count, CharT ch ); (since C++20)
basic_string& insert( size_type index, const CharT* s ); (until C++20)
constexpr basic_string& insert( size_type index, const CharT* s ); (since C++20)
basic_string& insert( size_type index, const CharT* s, size_type count ); (until C++20)
constexpr basic_string& insert( size_type index, const CharT* s, size_type count ); (since C++20)
basic_string& insert( size_type index, const basic_string& str ); (until C++20)
constexpr basic_string& insert( size_type index, const basic_string& str ); (since C++20)
basic_string& insert( size_type index, const basic_string& str, size_type s_index, size_type count ); (until C++14)
constexpr basic_string& insert( size_type index, const basic_string& str, size_type s_index, size_type count ); (since C++14)
basic_string& insert( size_type index, const basic_string& str, size_type s_index, size_type count = npos ); (until C++20)
constexpr basic_string& insert( size_type index, const basic_string& str, size_type s_index, size_type count = npos ); (since C++20)
iterator insert( iterator pos, CharT ch ); (until C++11)
iterator insert( const_iterator pos, CharT ch ); (since C++11)
constexpr iterator insert( const_iterator pos, CharT ch ); (until C++20)
void insert( iterator pos, size_type count, CharT ch ); (since C++11)
iterator insert( const_iterator pos, size_type count, CharT ch ); (until C++20)
constexpr iterator insert( const_iterator pos, size_type count, CharT ch ); (since C++20)
template< class InputIt >
void insert( iterator pos, InputIt first, InputIt last ); (until C++11)
template< class InputIt >
iterator insert( const_iterator pos, InputIt first, InputIt last ); (since C++11)
template< class InputIt >
constexpr iterator insert( const_iterator pos, InputIt first, InputIt last ); (since C++20)
iterator insert( const_iterator pos, std::initializer_list<CharT> ilist ); (since C++11)
constexpr iterator insert( const_iterator pos, std::initializer_list<CharT> ilist ); (until C++20)
template< class StringViewLike >
basic_string& insert( size_type index, const StringViewLike& t ); (since C++17)
template< class StringViewLike >
constexpr basic_string& insert( size_type index, const StringViewLike& t ); (until C++20)
template< class StringViewLike >
basic_string& insert( size_type index, const StringViewLike& t, size_type t_index, size_type count = npos ); (since C++17)
template< class StringViewLike >
constexpr basic_string& insert( size_type index, const StringViewLike& t, size_type t_index, size_type count = npos ); (until C++20)
```

PageDiscussionStandard revision: C++23ViewEditHistory

C++Strings librarystd::basic_string

std::basic_string<CharT,Traits,Allocator>::insert

```
constexpr basic_string& insert( size_type index, size_type count, CharT ch ); (1)
constexpr basic_string& insert( size_type index, const CharT* s ); (2)
constexpr basic_string& insert( size_type index, const CharT* s, size_type count ); (3)
constexpr basic_string& insert( size_type index, const basic_string& str ); (4)
constexpr basic_string& insert( size_type index, const basic_string& str, size_type s_index, size_type count = npos ); (5)
constexpr iterator insert( const_iterator pos, CharT ch ); (6)
constexpr iterator insert( const_iterator pos, size_type count, CharT ch ); (7)
template< class InputIt >
constexpr iterator insert( const_iterator pos, InputIt first, InputIt last ); (8)
constexpr iterator insert( const_iterator pos, std::initializer_list<CharT> ilist ); (9)
template< class StringViewLike >
constexpr basic_string& insert( size_type index, const StringViewLike& t ); (10)
template< class StringViewLike >
constexpr basic_string& insert( size_type index, const StringViewLike& t, size_type t_index, size_type count = npos ); (11)
```

Ah, that's good.

Useful tools

- [cplusplus reference](#): another document website, but easier to understand. Unfortunately, the documents stop at C++14 standard.
- [C++ standard](#).
 - Though I don't encourage you to be a language lawyer, sometimes it's important to refer to the standard to get the answer...
- [Compiler Explorer](#): an online code editor which can show assembly code & execute code, so that you can analyze code behaviors.
 - You can also choose different compilers & options too, so that you can try newest features without a e.g. virtual machine.

Useful tools

- [cpp insights](#): convert your code to a version without syntax sugar (语法糖) .
- [Quick Bench](#): an online benchmark website
 - You can test code efficiency & performance-consuming positions with different compiler options.
- Markdown: not for C/C++, just a really light-weight markup language for a notebook.
 - Code block, table, paragraph, etc.
 - Better than Word because code highlight & easy format.
 - Widely used in zhihu, stackoverflow, etc.
 - Many softwares for rendering md; Search them yourself!

显然，调用函数之前其参数一定evaluation完毕。但特别注意，参数之间的evaluation是没有顺序的，例如：

```
int f(int b, int c)
{
    static int a = 0;
    a += b + c;
    return a;
}

int g(int b, int c)
{
    return b - c;
}

int b = g(f(0, 1), f(1, 2));
// can be as : f(0, 1) => 1; f(1, 2) => 4; g(1, 4) => -3.
// or : f(1, 2) => 3; f(0, 1) => 4; g(4, 3) => 1.
```

Some learning materials

- [知乎](#)：手机上看的话，直接跳转到知识区或者科学区；电脑上就不要刷了，可以用来搜搜东西、看看文章。
 - 有非常多C++大佬：[吴咏炜](#)，[Mick235711](#)，[Lancern](#)...（至少二十几个，不胜枚举，大家刷刷C++话题就能找到很多）。
- [思否](#)：更适合中国宝宝体质的stackoverflow.
- 哔哩哔哩
- [博客园](#)
 - 由于博客园经费紧张，因此不让百度爬虫，百度因此给博客园降权，所以用百度搜索问题基本搜不到博客园的回答；去官网的搜索栏搜索。
- Cpp con
 - Though it's originally uploaded to YouTube, you can check it in [bilibili](#).
 - Great thanks to rSkip!!!!

Introduction

Concepts need to be known

C++ committee

- How does C++ standard evolve to the next version?
 - C++ standard is ISO standard (**ISO/IEC JTC1/SC22/WG21**).
 - Since C++11, new version of standard is released every 3 years.
 - That's because C++98 to C++11 takes too much time with too many features, which makes C++ loses lots of developers and not easy to learn.
 - So since C++11, train-like model is used, i.e. if a feature hasn't been adopted when the train departs, then it has to wait the next standard.
- WG21 is divided into CWG (Core working group), LWG (Library WG), EWG (Evolution WG), LEWG (Library Evolution WG).
 - Everyone, no matter whether he/she is one of ISO/C++ members, can make suggestions on C++ by **proposal**.
 - Like you want a new standard library (send to LEWG), a new language feature (send to EWG), etc. See [here](#) for how to send a proposal to C++ committee.

C++ committee

- Members will then vote for/against proposals, and when the proposal is in favor by the major, it's accepted.
 - Otherwise it needs to submit **revision**, and revision will be reviewed again.
 - So a proposal is always like `PxxxxRy`.
 - If a proposal is a fix of historical problem, which means it does draft revision, then it may be also marked as `DRxx`.
 - E.g. DR20 will be seen as part of C++20, so on gcc (of high version, i.e. has implemented such revision) `-std=c++20` will enable it though it's proposed in C++23.
 - You can check proposals and their vote results [here](#).
- There are also many searching groups (SG), who tries to explore new libraries or language features.
 - Like modules is SG2, concept is SG8, and after C++20 they've been disbanded (任务结束, 自行解散) .

For more information about C++ history, see B.S.'s paper [Thriving in a Crowded and Challenging World: C++ 2006-2020](#) on HOPL4 in 2020 ([Chinese version](#)).

C++ implementation

- C++ committee only provides a standard, and implementations of compilers and standard libraries are updated by vendors.
 - But the committee isn't just theorist; Google, Meta, Microsoft, etc. all have many important members in the committee.
 - The Big 3:
 - GNU: [gcc/libstdc++](#), widely used in Unix-based systems.
 - It's just mirror since gcc is created before Github.
 - LLVM: [clang/libc++](#), widely used in MacOS (gcc on Mac is in fact Apple-Clang, a changed version of Clang). `-stdlib=libc++`.
 - Microsoft: [msvc/MS-STL](#); msvc is not open-source. Widely used in Windows.
 - Usually Microsoft implements libraries faster, while msvc updates slower (i.e. slow language feature update), possibly because it's not open-source.
 - And there are many other vendors, like Intel's icpx for HPC, EDG for C++ frontend (Visual Studio Intellisense uses EDG), nvcc for CUDA, etc.

C++ evolvement

- C++ evolves slowly but continuously...
 - Some are arguing: C++ evolves too slowly!
 - Others: C++ is too complex!
 - C++ tries to keep a subtle balance, so that only necessary new features are introduced.
- And a final question: C/C++
 - Is C subset of C++?
 - Long before yes (C++98 v.s. C89), now no.
 - It's even not necessary to learn C first.
 - Is C always faster than C++?
 - Still no, because compilers have more optimizations on C++.
 - But if you don't understand modern C++, it's easy to write garbage code...



c.yy

我就期待别添加新特性了，他妈现在出去面试完全没信心了

07-03 · IP 属地四川



James Yin

本来期待executor，已经没有了

06-30 · IP 属地北京

Undefined behavior

- 一个幽灵，一个UB的幽灵，在C++的世界里回荡...
 - To keep efficiency, many things are required to be checked by C++ programmers, instead of by the language.
 - Undefined behavior means “such things should be prohibited, because anything may happen”.
 - For example, **signed** integer overflow (like `INT_MAX + 1`) is UB, since different hardware platforms have different handlers (like just overflow, or throws an hardware exception, etc.).
 - “Anything may happen” means that compiler may do any optimization or even strip the code out.
 - E.g. if the compiler has already known signed integer `a > 0 && b > 0`, then `bool c = (a + b) > 0` will be optimized to `bool c = true;` without checking overflow.
 - We’ll teach you many UBs, and some of them may cause very astonishing results.

Implementation-defined behavior

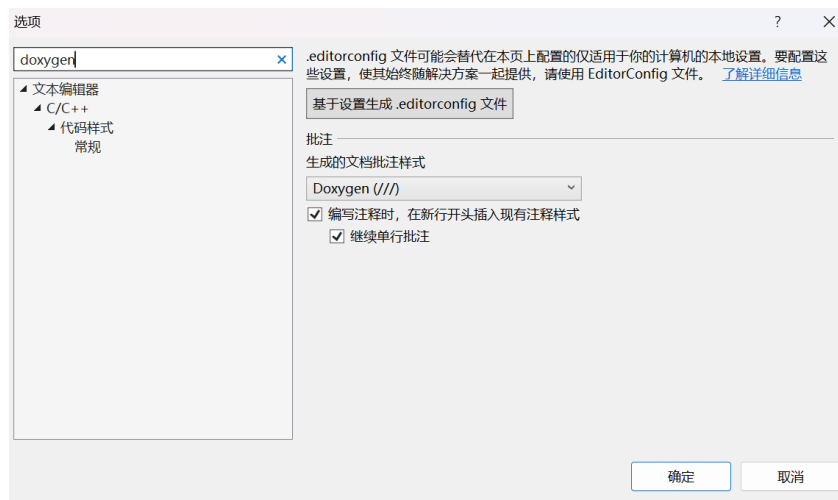
- Compared with UB, some behaviors are not determined by the standard, but are implementation-defined.
 - That is, it should be regulated like in the document of the platform.
 - For example, `sizeof(long)` is 4 in many Windows system, 8 in many Linux system.
 - So, if your C++ code runs on the specific platform, you may utilize such features.

Introduction

Code Style Suggestions

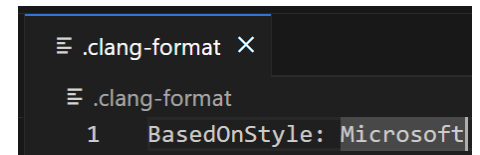
Code Style

- C++ doesn't have a uniform code style, so personally,
 - Name convention:
 - I used to code C# for a period of time, so I prefer to name namespace, class, function with upper camel case, and variables with lower camel case.
 - For example, `void SomeFunction(int someVariable);`
 - For private or protected members of a class, I'll decorate with a trailing `_`, e.g. `int privateMember_;`
 - Make your name good to understand without comment.
 - Sometimes it's necessary to give comments on class, methods, etc., and I'll use doxygen-style.
 - In VS, tool -> option ->
 - `// TODO: xxx` to mean your future plan.



Code Style

- You can have your own name convention, but there are two prohibition:
 - 1. Never begin with underscore (unless the name is just `_`).
 - `_` is usually used to denote “discarded value”, and since C++26 it can be defined more than once in a single scope.
 - For example, `std::lock_guard_{ mutex };.`
 - 2. Never have two underscores connected.
 - These two conventions are reserved by the standard library; such names may be conflict with them.
- For other styles, you can use ***clang format*** to check it!
 - I prefer [Microsoft code style](#) with slight changes; I may share it in code examples.
 - You need to give a `.clang-format` at root path of the project.

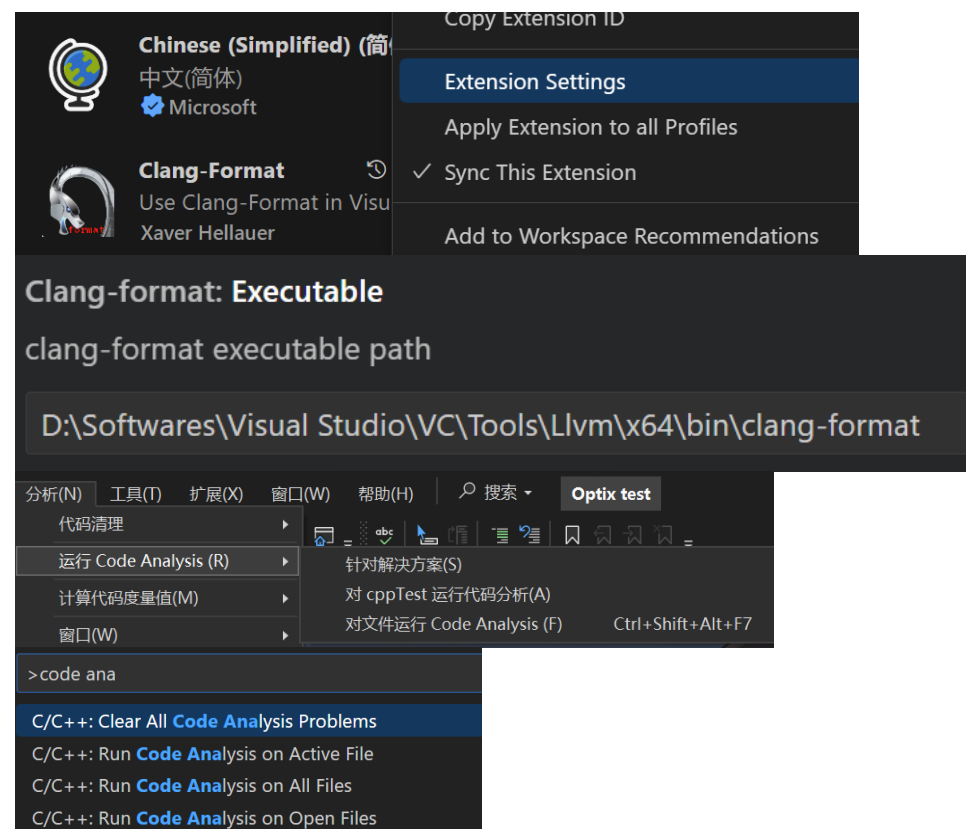


Code Style

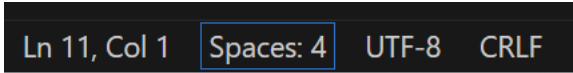
Ctrl + Shift + P

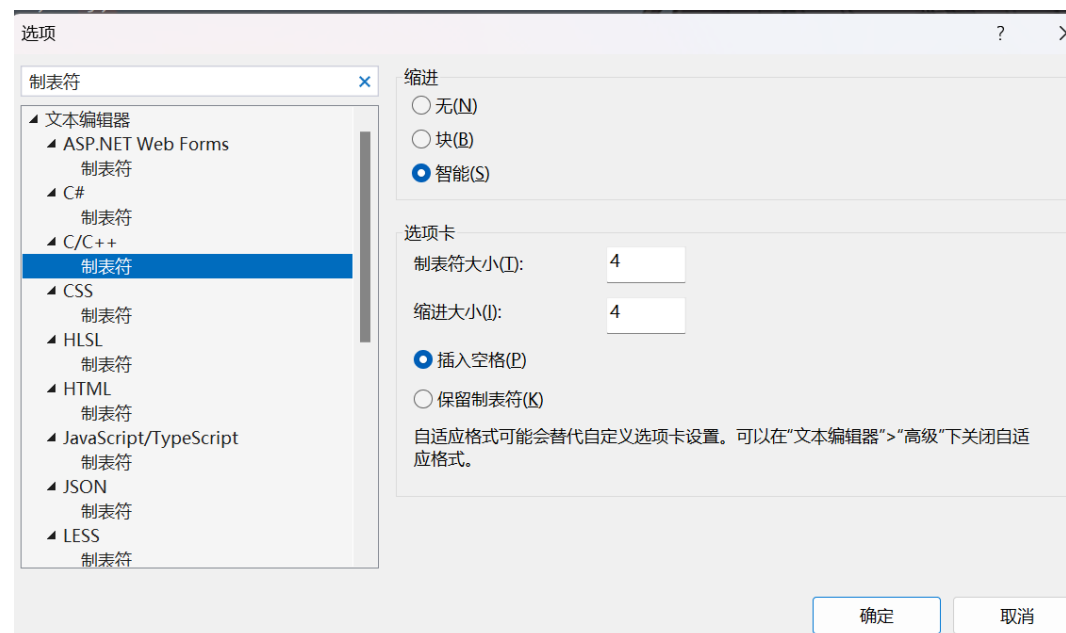
```
>Pre
Preferences: Open User Settings (JSON)
{
  "[cpp]": {
    "editor.defaultFormatter": "xaver.clang-format",
    "editor.formatOnSave": true
  },
}
```

- For VSCode, download plugin “Clang-Format”, and manage your settings.
 - Set your executable path to the one owned by Visual Studio.
 - On Linux/Mac, install manually, e.g. `apt install clang-format`.
- Besides, you may also utilize code analysis to check code quality.
 - For example, `if(a = 1)` is likely to be `if(a == 1)`, and code analysis will warn you.



Code Style

- Another important thing is Tab size...
 - Personally I like to type Tab to mean a new segment, but it has different length on different platforms (like 4 or 8), so the code will be messy.
 - You should change your settings on IDE, like in VS:
 - Tool -> option -> tick "insert spaces", and choose your size of Tab.
 - VSCode can adjust it at the right bottom.

- Clang format will also set the tab size; LLVM may be 2 by default.
 - If you hope to override some rules, you can check it [here](#).
 - For example, add `IndentWidth: 4`.



Next lecture...

- We'll review what you should have learnt before, and make some miscellaneous extensions.