# 1.演进、环境与资源

C++ standard之演化

C++98 (1.0)

C++03 (TR1)

C++11

C++14

C++标准库的头文件不带.h ，例如 #include <vector>

新式C头文件不带.h， 例如#include <cstdio>

旧式C头文件带.h，例如#include <stdio.h>

标准库的东西都放在命名空间std中

using namespaces std;

std::

许多TR1的特性在std::tr1的命名空间里，现在已经全部存在std命名空间里

编译器是否支持C++2.0?

<https://isocpp.org/blog/2014/03/compiler-support-for-c11-and-c14>

编译器

Language standard: 选择ISO C++11 国际标准的C++11

重要网页：

cpluscplus.com

<https://en.cppreference.com/w/>

gcc.gnu.org

书籍：

《C++ Primer》

《The C++ programming language》

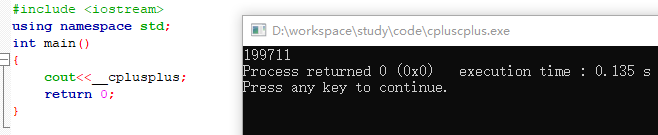
《The C++ standard library》

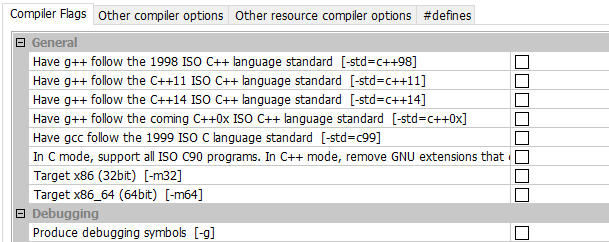
《Effective Modern C++》

全文检索工具

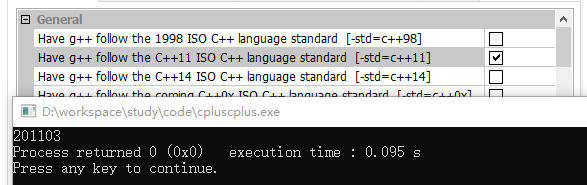
Windows grep

第一步，确认支持C++11

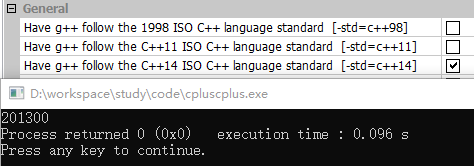




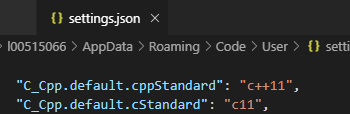
选择C++11

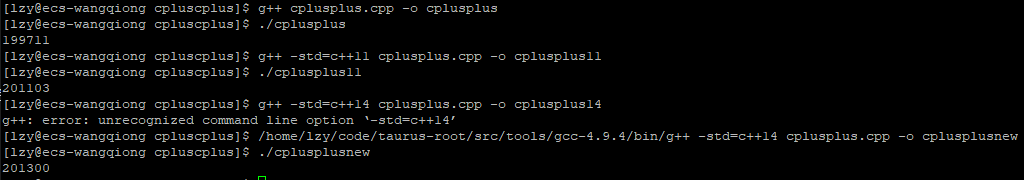


选择C++14



VSCode





# 2. Variadic Templates

可变参数模板

void print() //为了处理边界

{

}

template <typename T, typename… Types>

void print(const T& firstArg, const Types&… args)

{

cout<<firstArg<<endl;

print(args…)

}

…表示可以接受任何可变参数,包括0个

可以帮助我们完成递归

这个例子表示：print的参数可以有任意个，可以是任意类型

print(7.5, “hello”, bitset<16>(377),42);

#include <iostream>

#include <bitset>

using namespace std;

void print()

{

}

template <typename T,typename... Types>

void print(const T& firstArg, const Types&... args)

{

cout<<firstArg<<endl;

print(args...);

}

int main()

{

print(7.5, "hello", bitset<16>(377),42);

return 0;

}

sizeof…(args)

…就是一个所谓的包（pack）

用于template parameters（模板参数），就是template parameters pack （模板参数包）

用于function parameter types，就是function parameter types pack（函数参数类型包）

用于function parameters，就是function parameters pack（函数参数包）

可以并存：

template<typename… Types>

void print(const Types&… args){}

哪个比较泛化，哪个比较特化？

recursive function call 递归函数调用

class CustomerHash{

public:

std::size\_t operator()(const Customer& c) const{

return hash\_val(c.fname,c.lname,c.no); // 调用的是1.

}

};

template<typename...Types>

inline size\_t hash\_val(const Types&... args) //1. 接受n个参数

{

size\_t seed=0;

hash\_val(seed,args...); //调用的是2.

return seed;

}

template<typename T, typename... Types>

inline void hash\_val(size\_t& seed, const T& val, const Types&... args) //2.接受1+1+n个参数，第一个参数必须是size\_t

{

hash\_combine(seed,val);

hash\_val(seed,args...); //调用的是2

}

template <typename T>

inline void hash\_val(size\_t& seed, const T& val) //3.接受1+1个参数。边界函数

{

hash\_combine(seed,val);

}

template <typename T>

inline void hash\_combine(size\_t& seed, const T&val) //4

{

seed^=std::hash<T>()(val)+0x9e3779b9+(seed<<6)+(seed>>2);

}

# 3. inTemplate Expression、nullType Deduction with auto