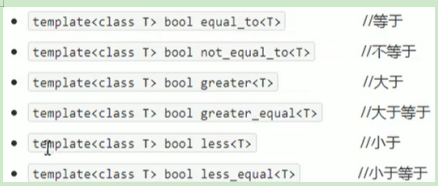
#<functional>:

1.内建函数对象（仿函数）:

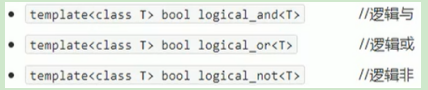
(1)算数仿函数：



(2)关系仿函数：可以在容器自动排序、STL算法中运用；



(3)逻辑仿函数：



2.function模板：可用于解决函数表问题（可以初始化函数或函数指针或lambda）。

#<algorithm>：

常用算法：

1.遍历：

（A）

其中\_func是函数或函数对象（遍历输出时用来**输出**）；

（B）

其中\_func是函数或函数对象（搬运时用于**返回值）**；

注：目标容器需要提前用resize开辟空间。

2.查找：

**！！！注意：find、find\_if、count、count\_if统计自定义数据类型时，需要重载operator== ！**

（A）元素查找：



（B）条件查找：

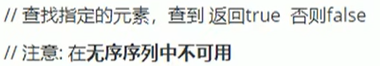




（C）相邻重复元素查找：



（D）查找元素是否存在（二分查找）：



（E）统计元素个数；

（F）条件统计；

3.排序：

（A）容器内元素排序；

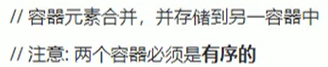




（B）打乱；

注：记得加上随机数种子

（C）合并容器；



注：目标容器需要提前开辟空间

（D）反转；

4.拷贝和替换:

（A）拷贝；





注：目标容器需要提前开辟空间。

（B）以旧换新；

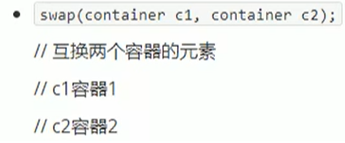


（C）





（D）

互换；



5.集合算法：注意**有序**和**开辟空间；**

（A）求交集；





目标容器开辟空间是从两个容器中**取较小值**；

（B）求并集；

目标容器开辟空间是从两个容器中**取和值**；

（C）求差集；

目标容器开辟空间是从两个容器中**取较大值**；

#<numeric>:

算术生成：

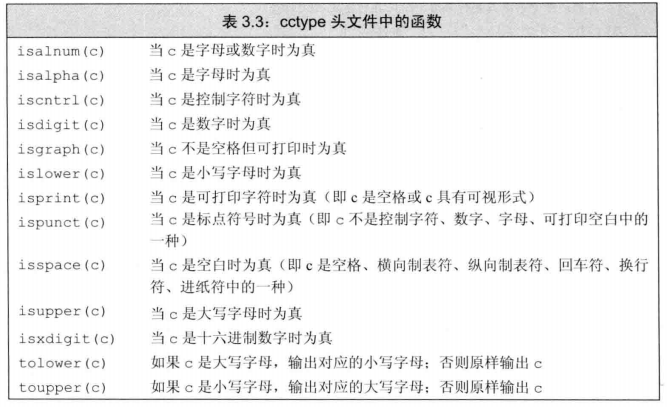
（A）累加；



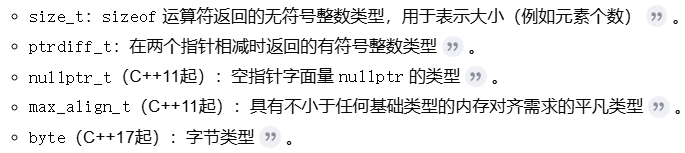
（B）填充；



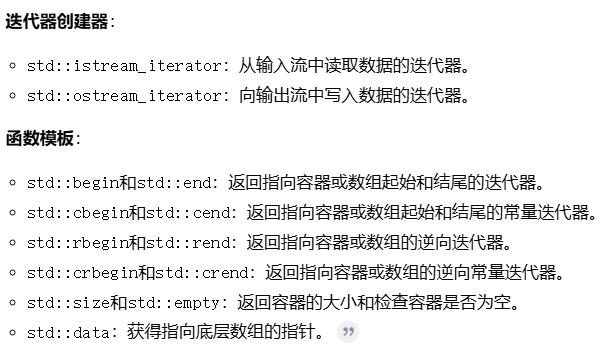
#<cctype>:



#<cstddef>:

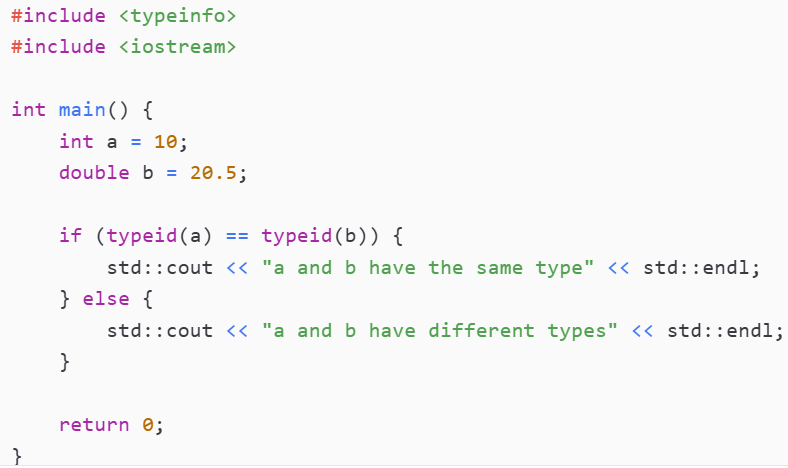


#<iterator>:



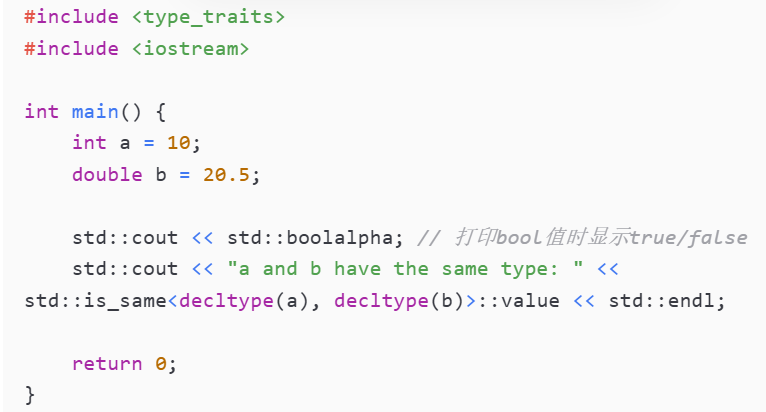
#<typeinfo>和<type\_traits>:

1.使用<typeinfo>的typeid函数；



typeid的name()可以显示类型。

2.使用<type\_traits>的is\_same模板（返回一个布尔值）：



输出true（如果类型不一样就输出false）。

对于自定义数据类型：

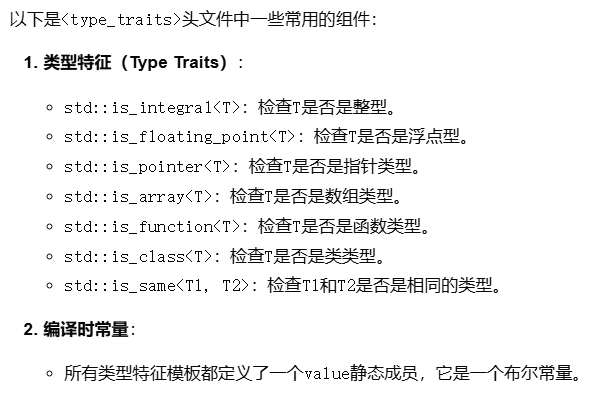


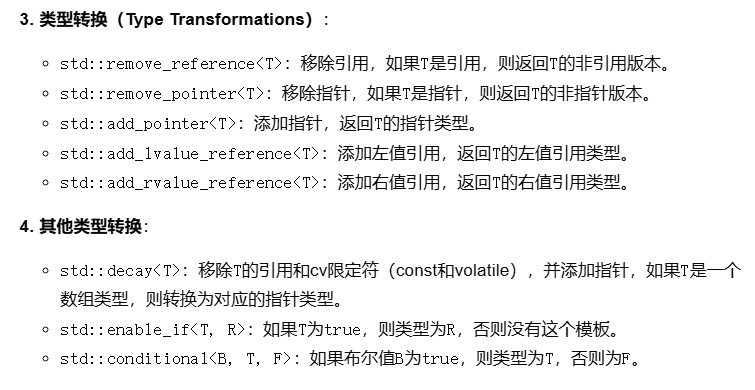
输出：



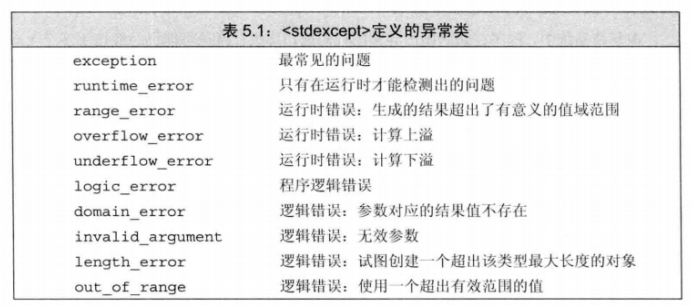
注：typeid适合运行时检查，而std::is\_same适合编译时检查。

3.<type\_traits>的类型判断和转换：





#<stdexcept>:



#<memory>:

智能指针：shared\_ptr<T>,unique\_ptr<T>,weak\_ptr<T>.

智能指针的动态数组：

1.std::get<T>(tuple)访问元素

2.std::tie<TTT>(a,b,c)=tuple<TTT>t;包装一个元组

3.auto t=std::make\_tuple(a,b,c);包装元组

4.std::tuple\_size<decltype(tuple)>::value元素数量，如：

*// 使用std::tuple\_size获取元组的大小*

constexpr size\_t tupleSize = std::tuple\_size<decltype(myTuple)>::value;

5.std::tuple\_element\_t<index,decltype(tuple)>获取index索引的元素类型，如：

*// 使用std::tuple\_element获取元组中第一个元素的类型*

using FirstElementType = std::tuple\_element\_t<0, decltype(myTuple)>;

std::cout << "第一个元素的类型是: " << typeid(FirstElementType).name() << std::endl;

*// 使用std::tuple\_element获取元组中第二个元素的类型*

using SecondElementType = std::tuple\_element\_t<1, decltype(myTuple)>;

std::cout << "第二个元素的类型是: " << typeid(SecondElementType).name() << std::endl;

6.std::apply(<可调用对象>,<tuple>);对元组成员进行某种操作，如：

*// 使用std::apply调用一个可调用对象*

auto print = [](auto a, auto b, auto c) {

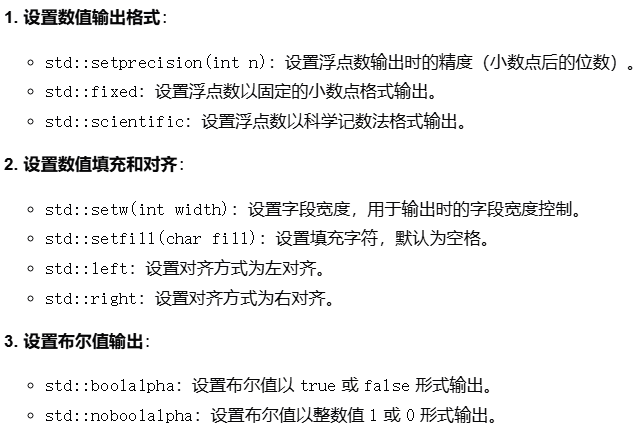
std::cout << a << ", " << b << ", " << c << std::endl;

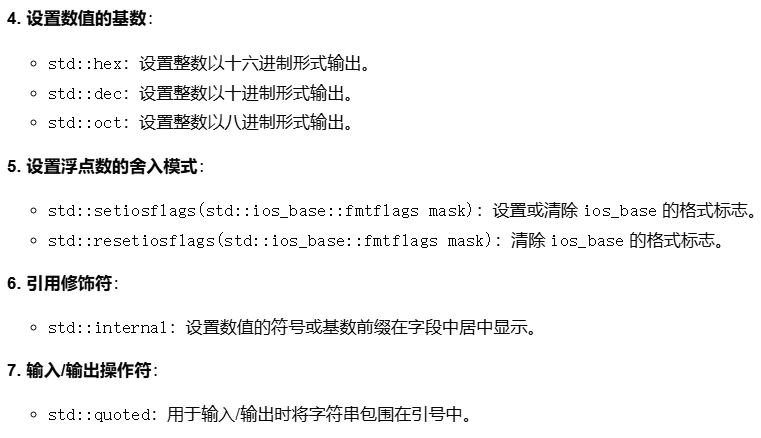
};

std::apply(print, myTuple);

#<iomanip>:

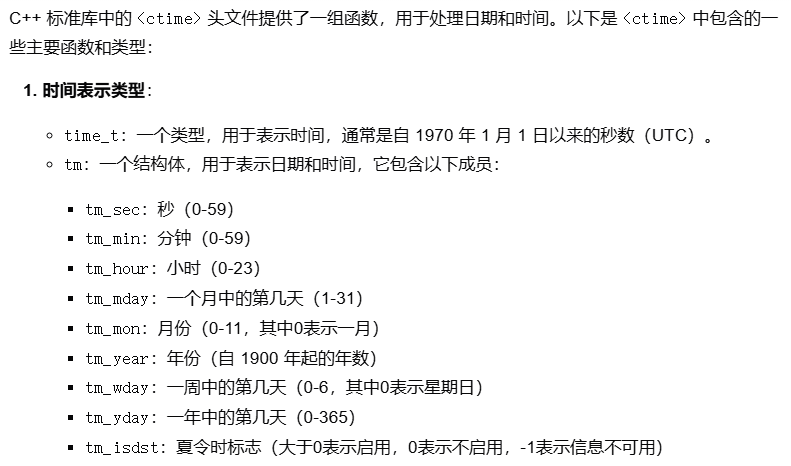
C++的format格式化工具，函数有：

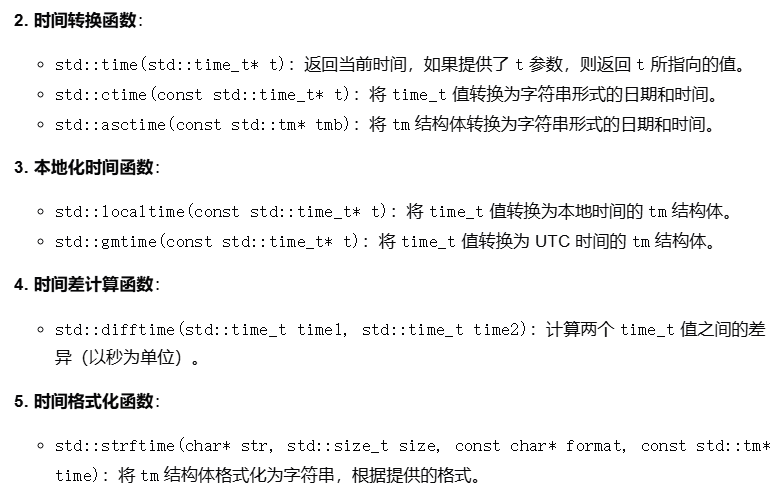






#<ctime>:



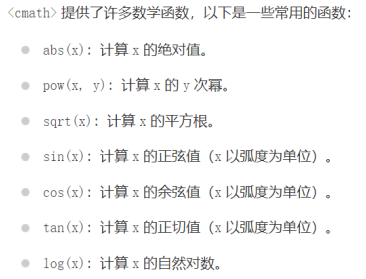


#<cstdlib>:



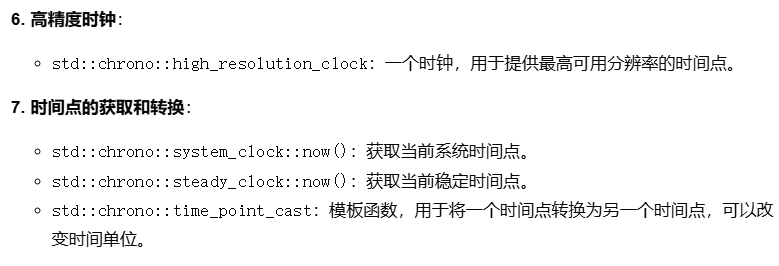


#<cmath>:



#<chrono>:





附：基于chrono写的加载条：

