## 程序调试

std::cerr << "getUserInput() called\n"; //用于调试代码

## 基本数据类型

无符号整数：当不需要负数（如物品的数量，数组索引）时，有双倍的表示范围

**unsigned** **short** us;

**unsigned** **int** ui;

**unsigned** **long** ul;

**unsigned** **long** **long** ull;

如果数学运算（例如算术或比较）有一个有符号整数和一个无符号整数，则有符号整数通常会转换为无符号整数。因此，结果将是无符号的。->应避免使用无符号数

固定宽度的整数：

***#include*** ***<cstdint>***

std::int#\_t

在存储整数值时，通常最好避免使用std::int8\_t和std::uint8\_t（以及相关的快速和最少类型），并改用std::int16\_t或std::uint16\_t。（因为8bit整形数通常被视作字符）

当输出浮点数时，std::cout的默认精度为6，使用std::setprecision() 重新设置精度。

ASCII字符：

‘A’=65,‘Z’=90,’a’=97,’z’=122

char ch{ 97};//使用 数字97，即字符 ‘a’ 初始化字符

char ch{‘5’};//使用字符5初始化

使用std::cin允许输入多个字符，但是ch只能容纳一个字符，多余的被暂存在缓冲区中

列表初始化可以在发生隐式类型转换时报错

显式类型转换：static\_cast<int>(5.5);

## 常变量，字符串

定义常变量时必须对其进行初始化，之后不能通过赋值更改该值

constexpr（“常量表达式”的缩写）变量只能是编译时常量。std::string、std::vector和其他使用动态内存分配的类型与constexpr不兼容

在编程时应拒绝使用意义不明的魔数，相反将其定义为一个常变量可维护性更强。

浮点字面值的默认类型为double而非float，若是想强制令其为float在末尾加上字段f，如**float** f { 4.1f };

**double** avogadro { 6.02e23 }; *// 6.02 x 10^23 是科学计数法写法*

所有的C字符串，都有一个隐式的null结尾符。例如"hello"，看起来只有五个字符，但实际是6个 ‘h’, ’e’, ’l‘, ’l’, ‘o’, and ‘\0’ (ASCII 码 0)。结尾的’\0’是一个特殊的字符，即是所说的null结尾符，用来标记字符串结束位置。

**int** x{ 012 }; *// 0 开头的数据在C++中代表八进制，应避免使用八进制数*

**int** x{ 0xF }; *// 0x 开头的数据在C++中代表十六进制*

可以使用一对（即两个）十六进制数字来精确表示整个字节。

**int** x{ 0b1 }; *// 0b 开头的数据在C++中代表二进制（c++14以后）*

**long** value { 2'132'673'462 }; *// 阅读起来更加容易，数字分隔符’只影响阅读体验，不会以任何方式影响对应的值。*

条件运算符（? :）是c++中唯一的三元运算符，可以帮助压缩代码，而不会丢失可读性。条件运算符是表达式而不是语句，因此可用于判断：

**constexpr** **int** classSize { inBigClassroom **?** 30 **:** 20 };

通常条件运算符用于二选一

inline关键字：内联函数（慎用）

C++20引入了关键字consteval，用于指示函数必须在编译时求值，否则将导致编译错误。这种函数称为即时函数（immediate functions）。

对string进行输入时，std::cin读取到 空白字符，就会停止，并将剩余内容存在缓冲区，规避使用getline结合ws使用。

std::string的返回长度不包括隐式null终止字符。

std::string::length() 返回无符号整数值（很可能是size\_t类型）。如果要将长度分配给int变量，则应将其static\_cast<int>

注意：初始化std::string的开销很大， 应尽量避免字符串的复制

using namespace std::string\_literals//方便使用s后缀

std**::**cout **<<** "goo\n"s; *//在双引号字符串后面使用s后缀来创建类型为std::string的字符串字面值。 s 后缀意味着 std::string 字面值*

std::string\_view（C++ 17）提供对现有字符串（C样式字符串、std::string或另一个std::string\_view）的只读访问，而不制作副本。只读意味着我们可以访问和使用正在查看的值，但不能修改它，大大减少了额外开销，像是看string的一个窗户

注：std::string\_view不会隐式转换为std::string，实在想用强制类型转换或用string\_view初始化一个新的string

将新字符串分配给std::string\_view会导致std::string\_view替换成新的字符串。但不会以任何方式修改先前的字符串。

与std::string不同，std::string\_view完全支持constexpr

视图查看的结果取决于正在查看的对象。如果正在查看的对象在视图在使用时被修改或销毁，将导致意外或未定义的行为。

不要使用std::string字面值来初始化std::string\_view。（字面值会在语句结尾自动销毁从而产生错误）

使用c样式字符串变量或字面值，std::string变量，std::string\_view变量或字面值均可。

修改std::string会使该std::string中的所有视图无效。

因为C样式的字符串在整个程序执行期间均有效（所以即使在某函数中，函数调用结尾依然不会销毁该串），因此可以从返回类型为std::string\_view的函数中返回C样式的串文本。

查看子字符串的能力带来了一个值得注意的后果，std::string\_view可以以null结尾，也可以不以null结束。注意不要编写任何假设std::string\_view以null结尾的代码。