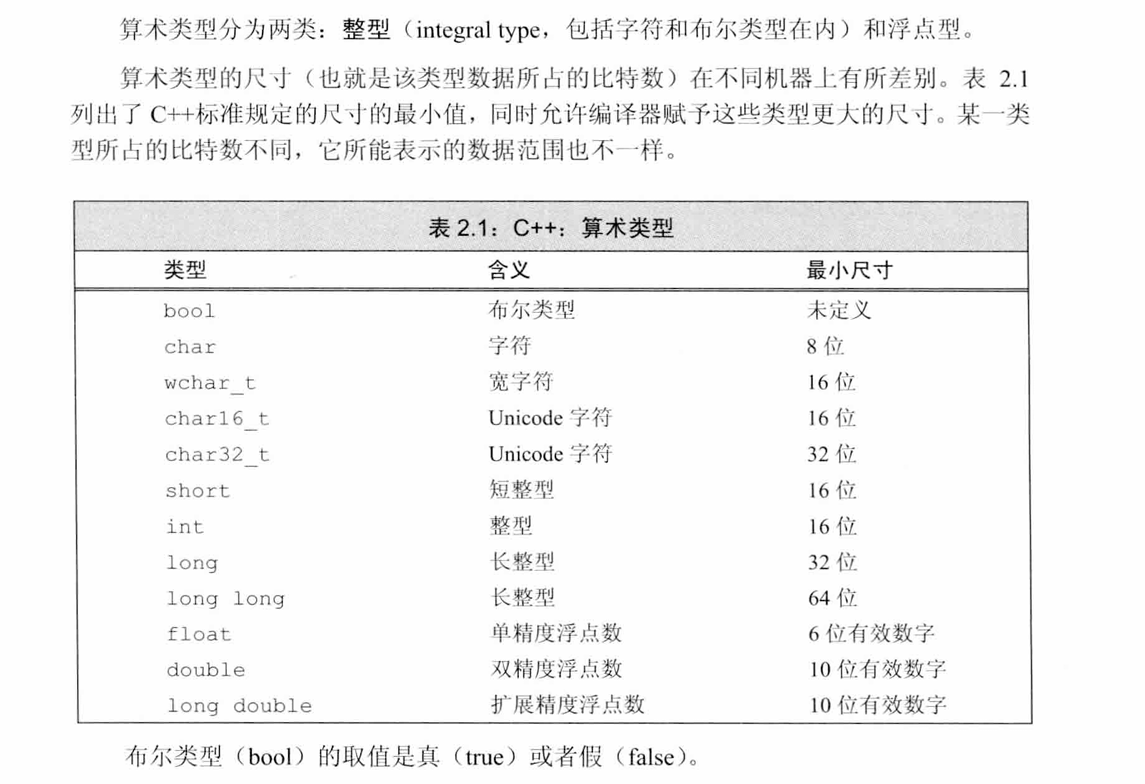
下表为C++中的算数类型及对应的在机器上的最小大小：



char 是否有符号由编译器决定，在编译时会使用 signed char 或 unsigned char 来指定是否有符号。

C++ 提供了几种字符类型：

1.char为一个 char 的空间应确保可以存放机器基本字符集中任意字符对于的数字值，即一个 char 的大小和一个机器字节一样。

2.wchar\_t为宽字符，用于扩展字符集，wchar\_t 确保可以存放机器最大扩展字符集中的任意一个字符（即两个字节）。

3.char16\_t 和 char32\_t为 Unicode 字符集服务。

编写C++代码时变量类型选用准则：

1.确定数值一定不为为负时，选用无符号类型。

2.​ 整数运算用 int，数值太大时用 long long，不用 short 和 long因为short范围太小而long通常和int的尺寸相同。

​3.浮点数运算用 double。float 和 double 精度差距大但计算代价相差无几。

4.不要用bool和char进行运算。因为char在不同机器上，是否有符号这件事是无法统一的。

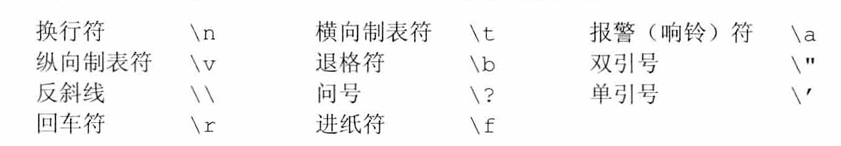
​类型转换：

1.把浮点数赋给整型时，结果仅保留小数点前的部分。

2.赋给无符号类型超出范围的值时，结果是初始值对无符号类型表示数值总数取模后的余数。

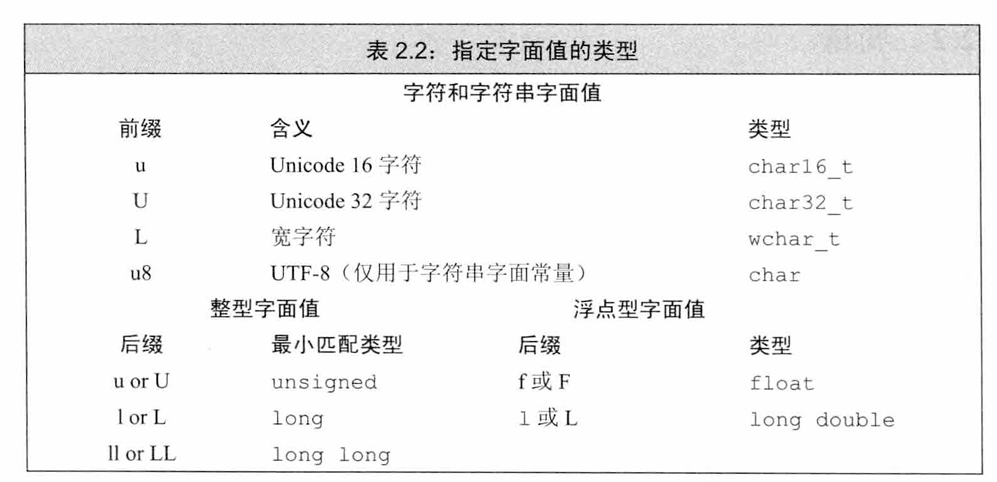
3.赋给带符号类型超出范围的值时，结果是未定义的。

C++中的转义符：



整型和浮点型字面值：整型字面值中 0 开头的整数是 8 进制，0x 开头的整数是十六进制。

字符和字符串字面值：单引号括起来的一个字符是 char 型字面值，双引号括起来的 0 个或多个字符则构成字符串型字面值。字符串字面值的类型实际上是字符数组，编译器会向每个字符串结尾添加一个空字符（‘\0’），因此字符串字面值的实际长度要比它的内容多 1。



ps：字面值的定义：在代码中直接表示的固定值

引用：

1.初始化引用时，是将引用和对象绑定在一起。引用无法重定向，只能一直指向初始值。引用是给对象起的别名。

2.引用必须初始化。引用的初始值必须是一个对象，不能是字面值。

3.对引用的所有操作都是对与之绑定的对象的操作。

4.引用不是对象。不能定义对引用的引用，因为引用并非对象。

5.引用只能绑定同类型对象。

指针：

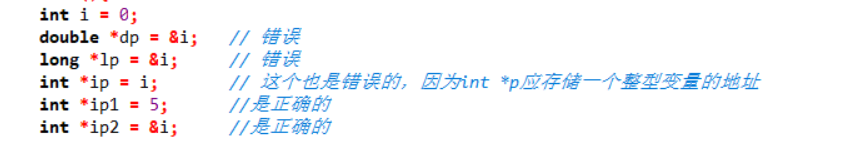
1.指针是一个对象；

2.指针可以重定向；

3.可以指向指针的指针；

4.指针不需要在定义时赋初值。

5.指针必须指向指定的类型，不能指向其他类型。



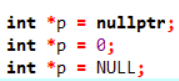
ps:解引用符（\*）的作用是访问指针指向的对象

不能定义指向引用的指针，但可以定义指向指针的引用：

image.png

ps：面对较复杂的指针或引用的声明语句时，从右向左读，这样容易弄清楚含义。

三种定义空指针的方式以及区别：



nullptr 是 C++11 引入的关键字，专门用于表示空指针。它的类型是 std::nullptr\_t，可以隐式转换为任何指针类型。而使用0赋值大多数情况下可以被隐式转换为指针，但表意并不如 nullptr 明确，可能会让读者产生误解。NULL 是在头文件 cstdlib 中定义的预处理变量，值为 0，也是一个整数常量，NULL 也可能导致类型不匹配的问题。

void\* 指针：void\* 指针是特殊的指针类型，可以存放任意对象的地址。

复合类型的声明包含一个基本数据类型和一组声明符（包含可能多个\*、&），类型说明符是声明的一部分。

const对象的声明和定义：

1.const对象必须初始化。

2.默认情况下const只在该文件内有效，如果需要在多个文件中共享一个const对象，必须在变量的定义前添加 extern 关键字并在本文件中声明。声明和定义都要加 extern，只需要在一个文件里定义常量，其他共享常量的文件里只需要声明即可。

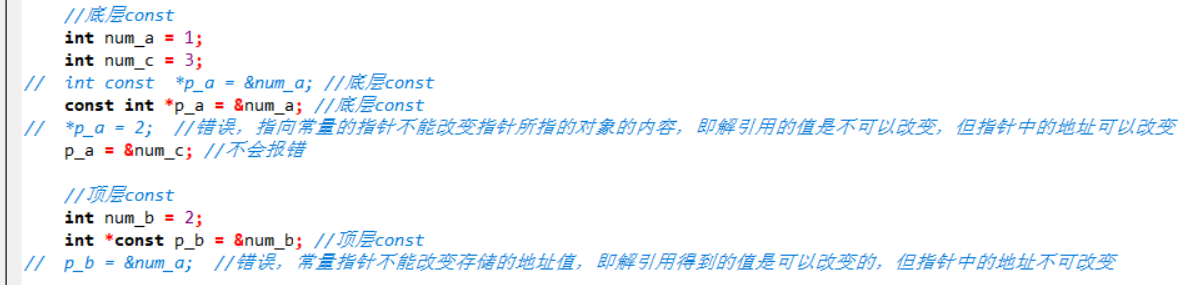
常量引用：创建一个引用，该引用指向一个常量。通过常量引用可以在不复制对象的情况下传递参数，同时确保在引用的上下文中不能修改该对象的值。

例如：

image.png

顶层cons（指针常量）和底层const（常量指针、指向常量的指针）：顶层 const 表示指针本身是个常量，底层 const 表示指针所指的对象是一个常量。

ps：对于指针和引用而言，顶层 const 在右边，底层 const 在左边。对于其他类型，全都是顶层 const**。**



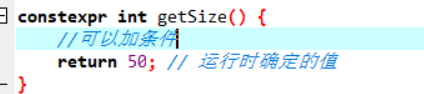
ps:执行对象的拷贝操作时，不能将底层 const 拷贝给非常量，反之可以，非常量将会转化为常量。

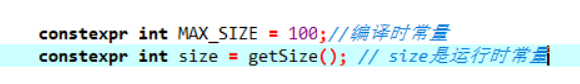
const和constexpr：

constexpr声明的变量必须用常量表达式初始化，限定了变量是编译器常量，即变量的值在编译器就可以得到。不能用普通函数初始化 constexpr 变量，但可以使用 constexpr 函数初始化 constexpr 变量。

const 未区分是编译器常量还是运行期常量。即 const 变量可以在运行期间初始化，只是初始化后就不能再改变了。因此const通常作只读使用。

ps：编译时常量是指在编译阶段就已经确定其值的常量。这意味着编译器在编译代码时就能够知道这个常量的具体值，并将其直接嵌入到生成的机器代码中。运行时常量是指在程序运行期间才能确定其值的常量。这意味着常量的值可能依赖于程序的输入或其他运行时条件，运行时常量的值在程序执行时被计算或获取。

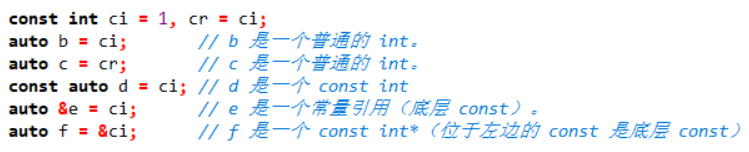




auto类型说明符：编译器根据初始值来分析表达式所属的类型（增加编译时间）

ps：auto 可以在一条语句中声明多个变量，但是多个变量必须是同一个基本数据类型（整型与整型指针和整型引用算一个类型）

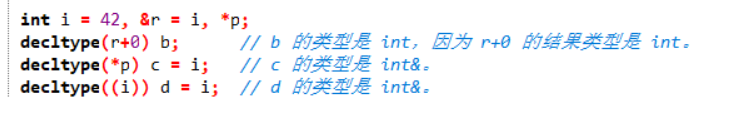
auto 会忽略引用与顶层 const（即引用和顶层const必须自己手动加）：



ps：如果初始化表达式是一个左值引用，auto会推导为相应的类型。如果初始化表达式是一个右值引用，auto会推导为相应的类型，但会去掉引用。

decltype类型说明符：当希望获得表达式的类型但不要值的时候使用decltype。

ps：decltype返回变量的类型包括顶层 const 和引用



ps：decltype((var)) 的结果永远是引用，而 decltype(var) 的结果只有当 var 本身就是引用时才为引用。使用关键字 decltype 时如果表达式的求值结果是左值，decltype 作用于该表达式（不是变量）得到一个引用类型。

auto和decltype的区别：

1.前者保留值，后者只获取类型

2.前者忽略顶层const和引用，后者不忽略