# 浮点数

编程设计过程中，我们经常需要处理具有小数的数据，例如身高，体重，距离，速度等等数据，单一的整形是无法满足我们的呢的设计需求的，所以，浮点数就应运而生。

# C++中的浮点数类型

Float

Double

Long double

# 有效位

C++中浮点数数据类型的取值范围决定于他们的数据有效位。

有效位：数字中有意义的数据位

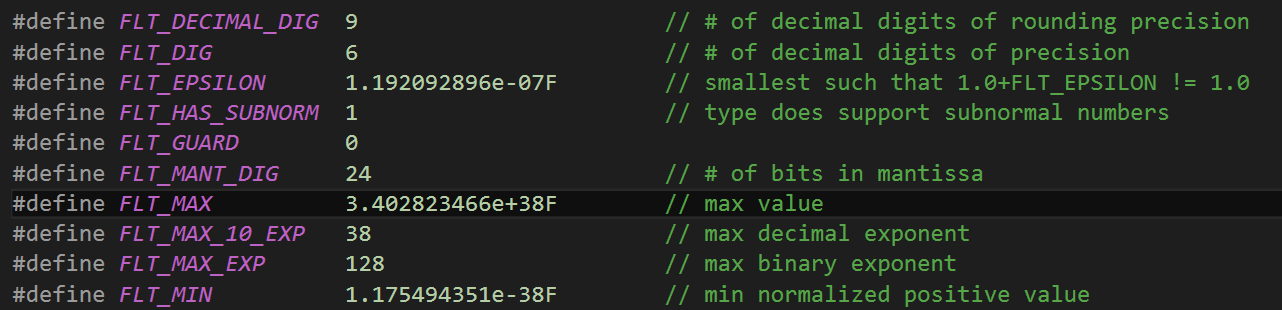
VS编译器中

Float的有效位是6位

Double的有效位是15位

Long double的有效位是15位

如果希望获知当前编译器中的浮点数有效位，可以查看头文件cfloat



# 设置cout输出小数的有效位个数

*cout*.*precision*(20);//设置有效位20个

有效的解决float和double输出有效位的问题。

# 如何在程序中标注浮点数常量

一般在c++中，对于小数的处理均按照double类型来处理。若果需要声明float类型的数据，需要进行数据标识，在数据末尾加缀f或是F。如果是long double，则可以在数据末尾标注L或是l，为了防止混淆，建议均使用大写的字母标注。

# 浮点数在程序中的书写方法

1. 直接借助小数点进行表述
2. E表示法（可以用来表述非常大或是非常小的数）

如果指数为正数，表示乘以10的E次方，负数表示除以10的E次方。书写注意不要在E之前和之后流空白

6.3E2 等于630

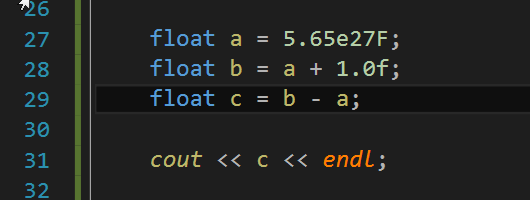
6.3E-2等于0.063

E的指数信息是有取值范围的，可以在cfloat头文件中查看到

# 浮点数的优缺点

浮点数弥补了整数与整数之间的数值空白，由于有缩放因子，可以表示的数值范围更大。缺点就是容易出现运算数据丢失，并且运算过程由于需要进行多次转码，运算速度慢，并且会伴随运算过程中精度丢失。

精度丢失的例子：



输出结果是0，而不是1，运算过程中精度出现了丢失。

# C++算术运算符

C++中提供了基本运算符

加 +

减 -

乘 \*

除 /

余 %

取于，也叫求模。a%b，则求出a与b相除的余数。

如果a大于b得出的余数肯定小于b。

如果a小于b，则计算结果等于a

# 组合运算符

C++中提供了更加方便的组合运算符，用来解决一个变量与另一个值进行计算，并将结果付给参与运算的变量。

+=

-=

\*=

/=

%=

# 类型转换

### 哪些情况？

1. 两种不同的类型数据进行赋值时，C++将进行数据类型转换。例如将整形赋值给浮点型。
2. 表达式中包含不同的类型是，C++将对值进行转换。例如运算
3. 参数传递时，类型不匹配可能会出现转换。

### 转换规则

只要相遇（预算，赋值），就往高处转，排序方式如下

Long Double

Double

Float

Longlong

Long

Int

Short

Char

### 强制类型转换

当进行数据匹配时，出现编译器警告，而可以编译通过时，我们可以显式的进行数据类型匹配要求，操作方式如下

float a = 10.0f;

double b = 2.0L;

int c = (int)a;

强制类型转换的原因

1. 为了解决计算过程预期值类型与提供的值类型不符的问题。例如不需要精确计算时，浮点类型可以转换为int类型数据，提供运算速度。
2. 为了解决转码问题，把char类型转换为int类型，进行输出。

# 运算符的优先级

先乘除余，再加减，同优先级运算符操作时，根据C++结合法则，从左到右依次计算。

# C++11中为我们提供了auto关键字

Auto关键字，自动类型推断符，让编译器能够根绝初始化的类型推断变量的类型。

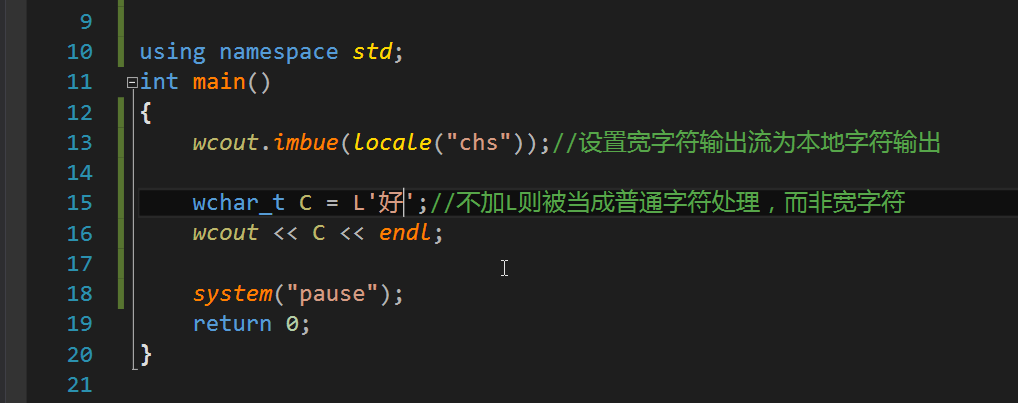
切记：auto关键字禁止在构建的时候不予以赋值。

# C++中的宽字符wchar\_t

由于char只能存储一个字节，用来储存更多字符时，显得无法操作。厂商为我们提供了新的存储宽字符数据类型wchar\_t，wt占用2个字节。

表示：

进行声明时，需要在字符内容前加入L



# C++11中提供的宽字符chart16\_t和char32\_t

进行初始化标记：

Char16\_t n = u’好’;//加小写的u进行标识

Char32\_t n = U’好’;//加入大写的U进行标识