[toc]

数据类型

C++ 内置了六种基本数据类型：

类型	关键字
布尔型	bool
字符型	char
整型	int
浮点型	float
双浮点型	double
无类型	void

布尔类型

一个 `bool` 类型的变量取值只可能为两种：`true` 和 `false`。

一般情况下，一个 `bool` 类型变量占有 1 字节（一般情况下，1 字节 =8 位）的空间。

字符型

`char` 类型的变量用于存放字符（实际上存储的仍然是整数，一般通过 ASCII 编码 实现字符与整数的一一对应）。`char` 的位数一般为 8 位。

一般情况下，`char` 的表示范围在 $-128 \sim 127$ 之间。

整型

int 类型的变量用于存储整数。

注意: `int` 类型的大小
在 C++ 标准中，规定 `int` 的位数 **至少** 为 16 位。
事实上在现在的绝大多数平台，`int` 的位数均为 32 位。

对于 `int` 关键字，可以使用如下修饰关键字进行修饰：

符号性：

- `signed`：表示带符号整数（默认）；
- `unsigned`：表示无符号整数。

大小：

- `short`：表示 **至少** 16 位整数；
- `long`：表示 **至少** 32 位整数；
- `long long`：表示 **至少** 64 位整数。

下表给出在 **一般情况下**，各整数类型的位宽和表示范围大小（少数平台上一些类型的表示范围可能与下表不同）：

类型名	位宽	表示范围
`short int`	16	$-2^{15}\sim 2^{15}-1$
`unsigned short int`	16	$0 \sim 2^{16}-1$
`int`	32	$-2^{31}\sim 2^{31}-1$
`unsigned int`	32	$0 \sim 2^{32}-1$
`long int`	32	$-2^{31}\sim 2^{31}-1$
`unsigned long int`	32	$0 \sim 2^{32}-1$
`long long int`	64	$-2^{63}\sim 2^{63}-1$
`unsigned long long int`	64	$0 \sim 2^{64}-1$

注意: "等价的类型表述" 在不引发歧义的情况下，允许省略部分修饰关键字，或调整修饰关键字的顺序。这意味着同一类型会存在多种等价表述。

> 例如 `int`, `signed`, `int signed`, `signed int` 表示同一类型，而 `unsigned long` 和 `unsigned long int` 表示同一类型。

单精度浮点型

`float` 类型为单精度浮点类型。一般为 32 位。

其表示范围在 -3.4×10^{38} 到 3.4×10^{38} 之间。

因为 `float` 类型表示范围较小，且精度不高，实际应用中常使用 `double` 类型（双精度浮点型）表示浮点数。

双精度浮点型

`double` 类型为双精度浮点型。一般为 64 位。

其表示范围在 -1.7×10^{-308} 到 1.7×10^{308} 之间。

无类型

`void` 类型为无类型，与上面几种类型不同的是，不能将一个变量声明为 `void` 类型。但是函数的返回值允许为 `void` 类型，表示该函数无返回值。

类型转换

在一些时候（比如某个函数接受 `int` 类型的参数，但传入了 `double` 类型的变量），我们需要将某种类型，转换成另外一种类型。

C++ 中类型的转换机制较为复杂，这里主要介绍对于基础数据类型的两种转换：数值提升和数值转换。

数值提升

数值提升过程中，值本身保持不变。

- `char` 类型和 `short` 类型在进行算术运算时会自动提升为 `int` 类型。类似地，`unsigned short` 类型在进行算术运算时会自动提升为 `unsigned int` 类型。
- 如果有必要（例如向一个接受 `long long` 类型参数的函数中传入 `int` 类型的变量），可以将位宽较小的整型变量提升为位宽较大的整型变量（注意符号性需保持不变，若符号性改变，则发生数值转换）。一个常见情况是：位宽较小的变量与位宽较大的变量进行算术运算时，会先将位宽较小的变量提升为位宽较大的变量。
- 位宽较小的浮点数可以提升为位宽较大的浮点数（例如 `float` 类型的变量和 `double` 类型的变量进行算术运算时，会将 `float` 类型变量提升为 `double` 类型变量），其值不变。
- `bool` 类型可以提升为整型，`false` 变为 0，而 `true` 对应为 1。

数值转换

数值转换过程中，值可能会发生改变。

- 如果目标类型为位宽为 x 的无符号整数类型，则转换结果可以认为是原值 $\bmod 2^x$ 后的结果。例如，将 `short` 类型的值 -1（二进制表示为 $1111\ 1111\ 1111\ 1111$）转换为 `unsigned int` 类型，其值为 65535（二进制表示为 $0000\ 0000\ 0000\ 0000\ 1111\ 1111\ 1111\ 1111$）。
- 如果目标类型为位宽为 x 的带符号整数类型，则 **一般情况下**，转换结果可以认为是原值 $\bmod 2^x$ 后的结果。[^note1]例如将 `unsinged int` 类型的值 $4\ 294\ 967\ 295$（二进制表示为 $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$）转换为 `short` 类型，其值为 -1（二进制表示为 $1111\ 1111\ 1111\ 1111$）。
- 位宽较大的浮点数转换为位宽较小的浮点数，会将该数舍入到目标类型下最接近的值。
- 浮点数转换为整数时，会舍弃浮点数的全部小数部分。
- 整数转换为浮点数时，会舍入到目标类型下最接近的值。
- 将其他类型转换为 `bool` 类型时，零值转换为 `false`，非零值转换为 `true`。

定义变量

简单地说，定义一个变量，需要包含类型说明符（指明变量的类型），以及要定义的变量名。

例如，下面这几条语句都是变量定义语句。

```
int oi;
double wiki;
char org = 'c';
```

在目前我们所接触到的程序段中，定义在花括号包裹的地方的变量是局部变量，而定义在没有花括号包裹的地方的变量是全局变量。实际有例外，但是现在不必了解。

定义时没有初始化值的全局变量会被初始化为 0。而局部变量没有这种特性，需要手动赋初始值，否则可能引起难以发现的 bug。

变量作用域

作用域是变量可以发挥作用的代码块。

全局变量的作用域，自其定义之处开始[^note3]，至文件结束位置为止。

局部变量的作用域，自其定义之处开始，至代码块结束位置为止。

由一对大括号括起来的若干语句构成一个代码块。

```
int g = 20;  // 定义全局变量
int main() {
  int g = 10;          // 定义局部变量
  printf("%d\n", g);  // 输出 g
  return 0;
}
```

如果一个代码块的内嵌块中定义了相同变量名的变量，则内层块中将无法访问外层块中相同变量名的变量。

例如上面的代码中，输出的 g 的值将是 10。因此为了防止出现意料之外的错误，请尽量避免局部变量与全局变量重名的情况。

常量

常量是固定值，在程序执行期间不会改变。

常量的值在定义后不能被修改。定义时加一个 `const` 关键字即可。

```
const int a = 2;
a = 3;
```

如果修改了常量的值，在编译环节就会报错：`error: assignment of read-only variable‘a’`。