第九章 常量

常量是一种标识符，它的值在运行期间恒定不变。C 语言用 #define 来定义常量（称为宏常量）。C++ 语言除了 #define 外还可以用 const 来定义常量（称为 const 常量）。

9.1 为什么需要常量

如果不使用常量，直接在程序中填写数字或字符串，将会有什么麻烦？

（1） 程序的可读性（可理解性）变差。程序员自己会忘记那些数字或字符串是什么意思，用户则更加不知它们从何处来、表示什么。

（2） 在程序的很多地方输入同样的数字或字符串，难保不发生书写错误。

（3） 如果要修改数字或字符串，则会在很多地方改动，既麻烦又容易出错。

- **【规则 9-1-1】** 尽量使用含义直观的常量来表示那些将在程序中多次出现的数字或字符串。

  例如：

```
#define         MAX    100       /*  C 语言的宏常量  */
const int       MAX = 100;       //  C++ 语言的 const 常量
const float     PI = 3.14159;    //  C++ 语言的 const 常量
```

9.2 const 与 #define 的比较

C++ 语言可以用 const 来定义常量，也可以用 #define 来定义常量。但是前者比后者有更多的优点：

（1） const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换时可能会产生意料不到的错误（边际效应）。

（2） 有些集成化的调试工具可以对 const 常量进行调试，但是不能对宏常量进行调试。

- **【规则 9-2-1】**在 C++ 程序中只使用 const 常量而不使用宏常量，即用 const 常量完全取代宏常量。

9.3 常量定义规则

- **【规则 9-3-1】**需要对外公开的常量放在头文件中，不需要对外公开的常量放在定义文件的头部。为便于管理，可以把不同模块的常量集中存放在一个公共的头文件中。

- **【规则 9-3-2】**如果某一常量与其它常量密切相关，应在定义中包含这种关系，而不应给出一些孤立的值。

  例如：

```
const   float    RADIUS = 100;
const   float    DIAMETER = RADIUS * 2;
```

9.4 类中的常量

有时我们希望某些常量只在类中有效。由于#define 定义的宏常量是全局的，不能达到目的，于是想当然地觉得应该用 const 修饰数据成员来实现。const 数据成员的确是存在的，但其含义却不是我们所期望的。const 数据成员只在某个对象生存期内是常量，而对于整个类而言却是可变的，因为类可以创建多个对象，不同的对象其 const 数据成员的值可以不同。

不能在类声明中初始化 const 数据成员。以下用法是错误的，因为类的对象未被创建时，编译器不知道 SIZE 的值是什么。

```
class A
{…
    const int SIZE = 100;  // 错误，企图在类声明中初始化 const 数据成员
    int array[SIZE];       // 错误，未知的 SIZE
};
```

const 数据成员的初始化只能在类构造函数的初始化表中进行，例如：

```
class A
{…
    A(int size);       // 构造函数
    const int SIZE ;
};

A::A(int size) : SIZE(size)    // 构造函数的初始化表
{
  …
}

A  a(100); // 对象 a 的 SIZE 值为 100
A  b(200); // 对象 b 的 SIZE 值为 200
```

怎样才能建立在整个类中都恒定的常量呢？别指望 const 数据成员了，应该用类中的枚举常量来实现。例如：

```
class A
```

```
{…
    enum { SIZE1 = 100, SIZE2 = 200}; // 枚举常量
    int array1[SIZE1];
    int array2[SIZE2];
};
```

枚举常量不会占用对象的存储空间，它们在编译时被全部求值。枚举常量的缺点是：它的隐含数据类型是整数，其最大值有限，且不能表示浮点数（如 PI=3.14159）。