**第十一章**

本章介绍了定义和使用类的许多重要方面。

一般来说，访问私有类成员的唯一方法是使用类方法。C++使用友元函数来避开这种限制。要让函数成为友元，需要在类声明中声明该函数，并在声明前加上关键字friend。

C++扩展了对运算符的重载，允许自定义特殊的运算符函数，这种函数描述了特定的运算符与类之间的关系。运算符函数可以是类成员函数，也可以是友元函数（有一些运算符函数只能是类成员函数）。要调用运算符函数，可以直接调用该函数，也可以以通常的句法使用被重载的运算符。对于运算符op，其运算符函数的格式如下：

operatorop(argument-list)

argument-list 表示该运算符的操作数。如果运算符函数是类成员函数，则第一个操作数是调用对象，它不在argument-list中。例如，本章通过为Vector类定义operator +( )成员函数重载了加法。如果up、right和result都是Vector对象，则可以使用下面的任何一条语句来调用矢量加法：

result = up.operator + (right) ;

result = up + right ;

在第二条语句中，由于操作数up和right的类型都是Vector，因此 C++将使用 Vector 的加法定义。

当运算符函数是成员函数时，则第一个操作数将是调用该函数的对象。例如，在前面的语句中，up对象是调用函数的对象。定义运算符函数时，如果要使其第一个操作数不是类对象，则必须使用友元函数。这样就可以将操作数按所需的顺序传递给函数了。

最常见的运算符重载任务之一是定义<<运算符，使之可与cout一起使用，来显示对象的内容。要让ostream对象成为第一个操作数，需要将运算符函数定义为友元；要使重新定义的运算符能与其自身拼接，需要将返回类型声明为 ostream&。下面的通用格式能够满足这种要求：

ostream & operator<<(ostream & os，const c\_name & obj)

{

os << . . . ; // display object contents

return os;

}

然而，如果类包含这样的方法，它返回需要显示的数据成员的值，则可以使用这些方法，无需在operator<<( )中直接访问这些成员。在这种情况下，函数不必（也不应当）是友元。

C++允许指定在类和基本类型之间进行转换的方式。首先，任何接受唯一一个参数的构造函数都可被用作转换函数，将类型与该参数相同的值转换为类。如果将类型与该参数相同的值赋给对象，则C++将自动调用该构造函数。例如，假设有一个String类，它包含一个将char \*值作为其唯一参数的构造函数，那么如果bean是String对象，则可以使用下面的语句：

bean = "pinto"; // converts type char \* to type String

然而，如果在该构造函数的声明前加上了关键字explicit，则该构造函数将只能用于显式转换：

bean = String("pinto"); // converts type char \* to type String explicitly

要将类对象转换为其他类型，必须定义转换函数，指出如何进行这种转换。转换函数必须是成员函数。将类对象转换为typeName类型的转换函数的原型如下：

operator typeName ( );

注意，转换函数没有返回类型、没有参数，但必须返回转换后的值（虽然没有声明返回类型）。例如，下面是将Vector转换为double类型的函数：

Vector: :operator double( )

{

. . .

return a\_double\_value;

}

经验表明，最好不要依赖于这种隐式转换函数。

与简单的C-风格结构相比，使用类时，必须更谨慎、更小心，但作为补偿，它们为我们完成的工作也更多。

**第十二章**

本章介绍了定义和使用类的许多重要方面。其中的一些方面是非常微妙甚至很难理解的概念。通常，对于诸如复制构造函数等概念，都是在由于忽略它们而遇到了麻烦后逐步理解的。本章介绍的一些内容乍看起来非常难以理解，但是随着经验越来越丰富，对其理解也将越透彻。

在类构造函数中，可以使 new为数据分配内存，然后将内存地址赋给类成员。这样，类便可以处理长度不同的字符串，而不用在类设计时提前固定数组的长度。在类构造函数中使用new，也可能在对象过期时引发问题。如果对象包含成员指针，同时它指向的内存是由new分配的，则释放用于保存对象的内存并不会自动释放对象成员指针指向的内存。因此在类构造函数中使用new类来分配内存时，应在类析构函数中使用dclete来释放分配的内存。这样，当对象过期时，将自动释放其指针成员指向的内存。

如果对象包含指向new分配的内存的指针成员，则将一个对象初始化为另一个对象，或将一个对象赋给另一个对象时，也会出现问题。在默认情况下，C++逐个对成员进行初始化和赋值，这意味着被初始化或被赋值的对象的成员将与原始对象完全相同。如果原始对象的成员指向一个数据块，则副本成员将指向同一个数据块。当程序最终删除这两个对象时，类的析构函数将试图删除同一个内存数据块两次，这将出错。解决方法是：定义一个特殊的复制构造函数来重新定义初始化，并重载赋值运算符。在上述任何一种情况下，新的定义都将创建指向数据的副本，并使新对象指向这些副本。这样，旧对象和新对象都将引用独立的、相同的数据，而不会重叠。由于同样的原因，必须定义赋值运算符。对于每一种情况，最终目的都是执行深度复制，也就是说，复制实际的数据，而不仅仅是复制指向数据的指针。对象的存储持续性为自动或外部时，在它不再存在时将自动调用其析构函数。如果使用new运算符为对象分配内存，并将其地址赋给一个指针，则当将delete用于该指针时将自动为对象调用析构函数。然而，如果使用定位new运算符（而不是常规new运算符）为类对象分配内存，则必须负责显式地为该对象调用析构函数方法是使用指向该对象的指针调用析构函数，方法是使用指向该对象的指针调用析构函数方法。C++允许在类中包含结构、类和枚举定义。这些嵌套类型的作用域为整个类，这意味着它们被局限于类中，不会与其他地方定义的同名结构、类和枚举发生冲突。

C++为类构造函数提供了一种可用来初始化数据成员的特殊语法。这种语法包括冒号和由逗号分隔的初始化列表，被放在构造函数参数的右括号后，函数体的左括号之前。每一个初始化器都由被初始化的成员的名称和包含初始值的括号组成。从概念上来说，这些初始化操作是在对象创建时进行的，此时函数体中的语句还没有执行。语法如下：

queue(int qs) : qsize(qs) , items(0) , frontNULL) , rear(NULL) { }

如果数据成员是非静态const成员或引用，则必须采用这种格式，但可将 C++11新增的类内初始化用于非静态const成员。

C++11允许类内初始化，即在类定义中进行初始化:

class Queue

{

private:

. . .

Node \* front = NULL;

enum {Q\_SIZE = 10);

Node \* rear = NULL;

int items = 0;

const int qsize = Q\_SIZE;

. . .

};

这与使用成员初始化列表等价。然而，使用成员初始化列表的构造函数将覆盖相应的类内初始化。

与简单的 C 结构相比，需要注意的类细节要多得多。作为回报，它们的功能也更强。