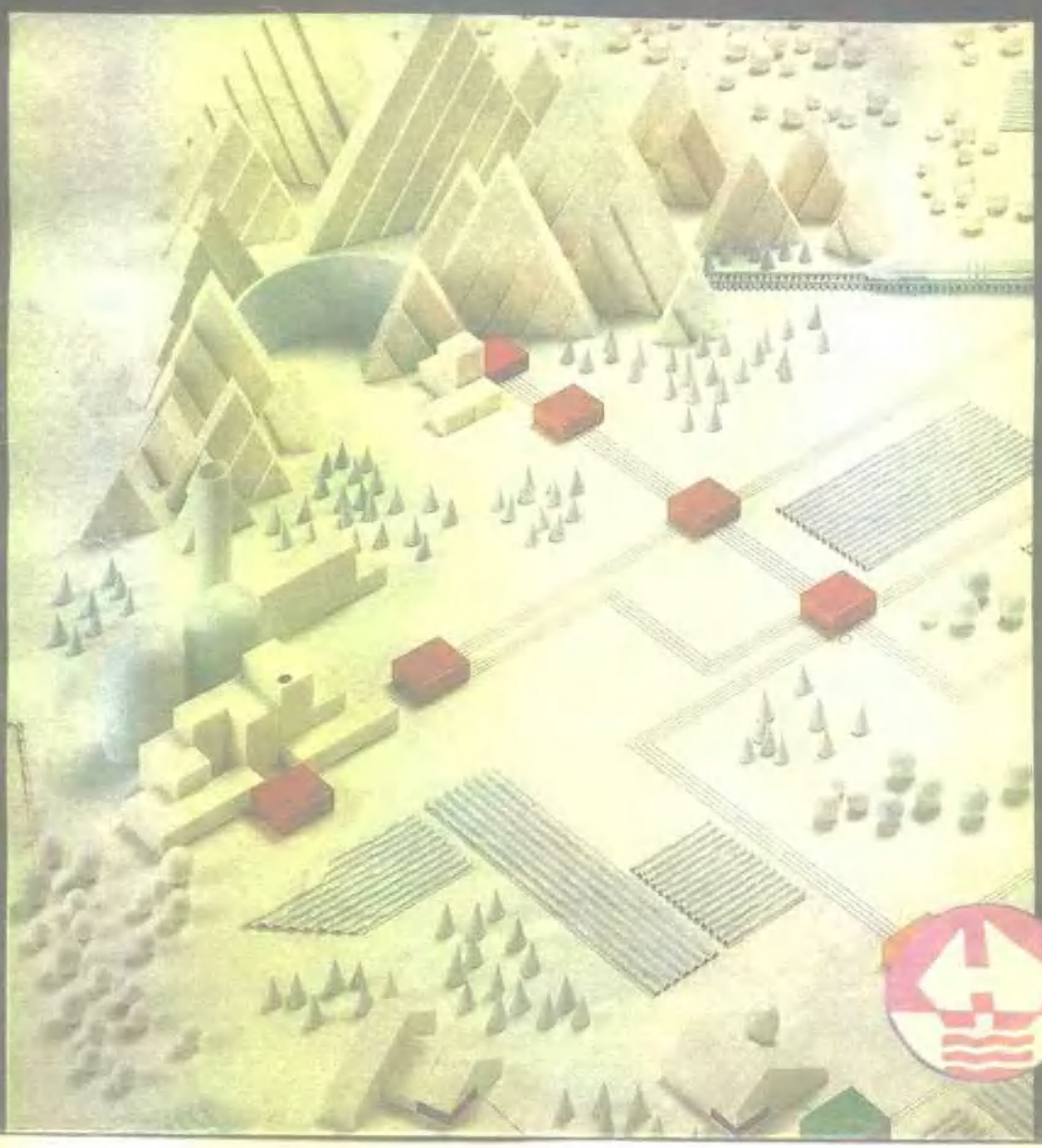
HOPE

C++编程教程

唐兰 编



北京希望电脑公司

TP312
T30

354189

C++编程教程

唐　兰　编译

北京希望电脑公司
一九九一年十月

前　言

C++已成为一种越来越受人们欢迎的程序设计语言,其吸引力不仅源于其父语言C之故，而且还在于其数据的抽象性以及面向对象的特征。在那些通过广泛的思想交流而对该语言的形成作出了贡献的用户之间具有积极意义的对话对该语言的发展起到了重要作用,其结果使得C++具有支持多种程序设计方法和技术的各种特征。

学习C++语言最好的方法是在书写程序过程中注意观察该语言的诸多特性是如何协调工作的。在该语言的形成过程当中，原作者同时作为C++用户以及C++实现者，从而有难得的机会在实现及使用C++的过程中看到为什么要开发某一语言特性并能了解如何对之进行改进。在这本书中，我们将一起来学习和评价作为程序设计工具的C++。

在原作者写本书的时候C++仍在不断地发展，到译者翻译的过程中C++正趋于发展的高潮。本书中尽量避免那些会使该语言其它版本的用户产生混淆的细节，但对诸如该语言的定义及实现的某些继承和提炼特征使得我们这里给出的C++与其它版本有所不同。本书的意图并非要写一本语言参考手册或语言的特定实现之描述。希望能够把用户从语言的特性细节引向程序设计的一般概念以帮助程序员有效地使用C++。

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第一章　介绍

C++是起源于C的通用程序设计语言。它在其父语言之上增加了大量的特性，其中最重要的支持数据抽象和面向对象的程序设计。C++保留了C的大多数特性，并延用了C的基本数据类型，操作，语法和程序结构。增加的特性使该语言类似的C的部分语言得以改善并能支持新的程序设计技术。

程序设计语言并不是自发地生成的，而是受人们探讨程序设计过程的方法之影响而产生的。这些思想经概括和处理之后达到一定规范，并建立起新的语言来支持它们。程序设计规范即是为程序设计和实现过程提供技术的模型。这些技术说明诸如一个设计如何实现一个程序设计问题，抽象的使用和程序的组成之类的问题。若一个语言的特性使语言很容易应用某种模型技术，则该语言支持该模型所对应的特定规范。

许多有关新程序设计语言的书都只是论及语言的特性而忽略了引导用户去探寻该语言的程序设计过程。对于象C++这样的语言，它的内容比用户可能熟知的其它许多语言都要丰富得多，因而上述方法并非可取。本书讨论如何用C++进行程序设计。我们将论及如何使用C++特性的细节，以及如何把规范用到程序的设计和实现中去。

本书力图介绍一些概念以使程序设计规范和语言特性能承上启下。尽管我们讨论了本语言的大多数特性，但并没有论及语言的每一个细节。我们给出的许多例子使用了UNIX操作系统函数而没有作详细的解释，但它们的应用是很直接了当的，以至不会防碍不熟悉UNIX系统的读者的理解。我们假设我们的读者是熟悉C的有经验的程序员，我们将直接详细讨论在用C++进行程序设计时会遇到的新内容。

1.1　C++语言

C++在C上增加的主要内容是类类型的引入。类允许用户定义集合数据类型，该类型不仅可以包含数据成员而且也可以包含操作该类型的函数。隐蔽在类中的数据提供数据抽象机制。类继承将数据抽象扩充至面向目标的程序设计。用户定义的操作符函数和转换使得类可以被加到预定义类型系统中去。　　类具有能使存贮管理嵌入动态数据结构的特征。本语言还提供通用动态分配和回收操作符。这就使得程序员能够修改存贮管理方案以适于特定的应用。

除了这些能支持建立数据结构的技术外，还具有能改进函数的使用之特征。所有C++函数声明必须包含参数类型信息。这因而支持了参数类型检查和函数重载。函数参数声明还可以包含没有在函数调用中给出的参数缺省值。C++中的引用类型支持通过引用传递参数，也支持值传递。

1.2　程序设计规则

程序设计规范是如何设计和实现程序的模型。不同的模型产生不同的技术。技术的不同并不隐含着它们之间是矛盾的，而且不同的技术可以看作相互弥补。程序设计模型间的共同之处在于设计应基于抽象这一概念，而抽象对应于程序设计问题中的各元素，并且实现应是一组模块的集合，最好是可重用模块之集。它们的区别在于如何构造该抽象以及什

么构成了一个模块。

较为成熟的过程程序设计方法是基于将程序作为一组函数集的模型。该技术提供了如何设计、组织和实现组成程序函数的指南。函数分解设计方法的特征是那些作为解决一个程序设计问题的抽象操作的函数。文件组织使得函数可被组织在分散的模块中;结构化程序设计技术则使得一个函数的实现易读易维护。

数据抽象着重于被面向过程技术忽略了的数据结构上。数据抽象模型即是数据结构应由对它进行的操作来定义，而不是由其实现的结构来定义。用于数据抽象的技术是把一个数据结构封闭在一个抽象数据类型中。对该结构的访问是通过作为该类型一部分的一组操作而提供的。数据抽象与把函数看作抽象操作的过程程序设计互补，因为若没有一个存在，则另一个也不会是完整的。

面向对象的程序设计基于把程序作为由一组抽象数据类型例而建立的模型。面向对象设计的特征是表示程序设计问题中对象的类型。在对象类型中的操作，象在过程程序模型中的函数一样，是解决问题的抽象操作。对象类型作为一个模块，它可以重用于解决相同领域内别的问题。

没有一种规范能适于解决所有的程序设计问题。程序设计需要专业知识，但它还不是一门科学。程序设计技术需要灵活地使用，注意思考它们是否适于手边要解决的问题。盲目使用当前最流行的规范并不能很好地对一个问题进行抽象。本书的主要目的是鼓励读者严格并灵活地思考程序设计和实现中的问题。

1.3 本书组织

我们打算教给读者如何用 C++设计并实现程序。我们着手于语言的数据类型和操作，并沿着数据结构和操作的组织到程序，最后讨论存贮管理和库设计这类更高一级的问题。

看待程序的一种方法是把它的描述成一系序列对数据的操作，C++数据类型和操作在第二章给出。C 采用的语言特性提供了抽象数据类型和操作，如数值和算术操作。用户定义类类型作为把数据类型和操作增加到语言中的方法而给出。

第三章讨论把操作系列组织到函数中的过程程序设计技术。功能分解和结构程序设计被当作设计规则讨论，此外还讨论了作用域和名字连接的支持语言特性，通过值和引用的参数传递，函数名重载，缺省变元以及插入函数。

第四章讨论 C++类的基本特性。该语言的直观表示奠定了书中以后各章讨论的基础。我们讨论数据和函数成员，构造器和解除器，操作符重载以及保护。

第五章讨论数据抽象类的应用。我们叙述了如何建立一个新数据类型，用类提供的功能使实现变得安全和可维护，以及如何定义把新类型融进现有类型系统的类型转换和操作。

第六章讨论类继承，该性质支持新类类型用现有类类型加以定义。继承可用于修改一个现有抽象数据类型，建立相关抽象类型的层次集合，组合无关类型特性以及提供函数调用的灵活的运行时间约束。

第七章把数据抽象和继承应用到面向对象的程序设计中。我们给出了将程序作为对象类型模块设计的指南，并说明动态的面向对象程序设计风格。

第八章给出 C++的内存管理特征。对于一般应用,一给定库或类类型或特定类型，都可开发并修改以得到有效的、可维护的存贮管理方案。

第九章我们讨论作为可重用软件模块集的库的设计和使用。我们给出如何设计库的方法，以使它们可扩展，可修改而不会影响库的源码。

在每一章的最后一节给出练习。其中一部分在附录中给出了答案。

我们把 C++语言的细节与软件设计和软件工种这种较大的问题联系起来。我们希望读者不仅学会如何有效地把 C++当作一个程序设计工具使用，而且还努力地用新的思维方式对程序设计和实现过程进行思考。

第二章　数据类型和操作

C++类型系统由基本语言定义类型，用户定义类类型和可以从基本和类类型导出的类型组成。本语言提供内部类型的操作和标准转换。类类型还可以有为之定义的操作和转换，因而这就允许类被当作预定义类型系统的一致延伸。

C++中的内部数据类型可理解成抽象的概念，如数或布尔值，或根据其计算中的表达方式，理解为位系列。具体如何理解依赖于用于处理不同类型值的操作。算术和逻辑操作符对类型作的是数学和逻辑理解。其它操作符对类型的理解依赖于位的表示。

2.1　数值类型

C++有整数类型和浮点类型。本语言提供被重载的算术操作符以处理整型值和浮点值，并定义数值类型间的转换。重载操作符即是那些用同样的符号表达一个以上不同操作实现的操作符。数值类型在与算术操作符一同使时被理解为数值表示。

下列程序中用了数值变量、数值值和算术操作符一起进行计算操作。用 int 表示整数，用 double 来表示实数。

```
#include <stdio.h>
main() {
/*
     Distance of a falling object from the point of its
     release at each of the first 10 seconds of its fall,
     in meters
*/
     const double g = 9.80;          // accelcration from gravity
     for( int t = 1; t <= 10; t++ ) {
          double distance = g * t * t / 2;
          printf( "\t%2d %7.2f\n", t, distance );
     }
}
```

本程序打印出一个物体开始降落的前十秒中在每一秒时所降落的距离：

```
1        4.90
2       19.60
3       44.10
4       78.40
5      122.50
6      176.40
7      240.10
8      313.60
9      396.90
```

10 490.00

计算距离的表达式中同时有 double 类型和 int 类型的操作数。其结果是一个 double 值，它存放在 double 变量 distauce 中。

```
double distance = g * t * t / 2;
```

尽管 t 和 2 都是 int 类型，但该计算过程完全是按浮点操作来执行的。因为 g 是 double 类型，在从左到右的乘除运算中，每一个操作符在计算过程中都至少有一个 double 类型的操作数，这使得 int 操作数进行转换以便与之匹配。

算术操作符被数值类型重载。操作数的类型决定做整型操作还是浮点操作。若表达式被修改以至按另外的次序完成各操作，则有一些操作的类型也可能改变。

```
distance = g * (t * t / 2);
```

用括号把整型操作数括起来，则 t 与 t 相乘为整型乘并产生一个 int 结果。这里的除法也作用于 int 操作数。该整型除产生一个 int 类型结果。当 t 为奇数时，结果的小数部分被丢掉，从而使该计算生成的结果与前面浮点操作生成的结果不同。若使其分母用浮点形式，该操作变成浮点除，其结果则与原来的结果相同。

```
distance = g * (t * t / 2.0);
```

对于一个重载操作符，操作数的类型影响选用操作符的那一种实现来执行操作。

算术操作符被定义为能对各种数值类型操作数进行操作。一元操作符有++，--，取负-和无操作数的+。二元操作符有*，/，+和-。另外，余数操作符%仅作用于整型操作数。

++和--操作符有修改作为其操作数的对象的值之付作用。它们可以用作前缀或后缀，但产生不同的结果。若作为前缀，表达式的结果是其对象作了增减之后的值。若作为后缀，表达式结果是对象作增减之前的值。其它算术操作符不修改其操作数的值。

一元操作符优先级高于二元操作符。乘法二元操作符*，/，和%的优先级高于加法二元操作符+和-。

整数类型有 char，short，int 和 loug。每一种类型都至少能够表示其前一种类型的值域，因而其每一种类型都可认为其长度与其前一种类型的长度相同或后者大于前者。整数类型可以有符号或无符号。在声明中，short，int 和 long 类型指示符都意指有符号的，除非显式地指定了 unsigned。当有不同类型的整数操作数同时出现在一个表达式中，它们将被转换成同一种类型。类型 char 和 short 的操作数总至少转换成 int。其它转换总是把操作数类型转换成比之长的类型。若一操作数类型为 unsigned 并长于或等于另一个操作数类型，则转换总是作用于 unsigned 类型。

浮点类型有 float，double，和 long double。每种类型都可以至少表示其前面一种类型的值，因而可认为每种类型都包含了前面一种类型。当有不同的浮点类型操作数同时出现在算术表达式中，则较长的操作数被转换成与其它操作数类型相同的类型。当有浮点操作数和整型操作数同时出现，则整型操作数被转换成与浮点操作数类型相同的类型。

任何数值值都能够赋给一个任何其它数值类型的对象，此时该值类型被转换成赋值符号左边对象的类型。

```
double d;
d = 42;
```

转换还可能会使值有部分丢失。

```
int i;
```

```
i = 3.1415;
char c;
c = 777;
```

在上面的式子中，i 得到值 3。值 777 可能过大而不能被表示在一个 char 类型对象中，因而经赋值后的 c 值是依赖于特定的实现而决定如何把长整数类型转换成短整数类型的。

转换操作符也可用来作显式转换。例如，在下面的式子中，count 的 int 值被显式地转换成 double。

```
int count = 1066, total = 1237;
double ratio;
ratio = double( count ) / total;
```

本例中用了一个函数调用转换符：

```
double( count )
```

而另一种同样的转换符操作形式为：

```
( double ) count
```

在简单的转换表达式中，这种函数形式通常被认为是一种清楚的转换方式。

数值值在程序中可直接用数值字符表示。浮点表示中可能有一个小数点和小数部分，以及还可能有指数：

```
9.80
0.98e1
98e-1
```

这些式子都表示同一个值并都具有 double 类型。对于具有各种浮点类型的值，用 L 或 l 指示 long double，F 或 f 指示 float。

```
0.98e1L
9.80f
```

整数数值具有 int 类型，除非它们的值过大或它们的类型由后缀说明。若一个整数数值太大而不能在一个 int 对象中表示，则它有可能有一个 long 类型。后缀 L 或 l 指示一个 long，U 或 u 指示一个 unsigned。这两个后缀可以合用，例如：

```
1642UL
```

八进制的表示方法是在第一个数字前放 0：

```
0777
```

十六进制的表示方法是在第一个数字前放 0x 或 0X：

```
0x1ff
```

数值值还可以在初始化值时用一个 const 来表示。

```
const double g = 9.80;
```

因为 const 指示 g 值是不可更改的，因而该标识符 g 总代表值 9.80。

整数值还可以用 enum 表的成员来表示。若没有进行过初始化，enum 的成员具有连续的 int 值，第一个成员为 0 值。

```
enum { mon, tues, wed, thur, fri, sat, sun };
```

这里，fri 的值为 4。当 enum 被显式地进行过初始化，未被初始化过的表中成员的值为在表中位于其前面的值加 1。

此为试读，需要完整PDF请访问：www.ertongbook.com

```
enum { mon = 1, tues, wed, thur, fri, sat = -1, sun };
```

在这种情况下，fri 的值为 5，而 sun 的值为 0。

字符集用可以被包含在一个 char 数据类型中的整数值表示。字符的表示方法是放在单引号中的字符或斜杠，这提供了字符值方便的表示方法：

```
char digit = '9';
char w = 'w';
char newline = '\n';
char tab = '\t';
char null = '\0';
```

字符值总是正的，即使它们被放在 signed char 中时也是正的。它们可以在表达式中当作整数类型操作数使用：

```
int value = digit - '0';
```

有效的字符值操作往往依赖于一定的标准字符集，如 ASCII 或 EBCDIC。

2.2 关系操作符和逻辑操作的标量类型

在 C++中没有布尔类型。标量类型，象整数类型，都作为布尔值使用，用零代表 FALSE，用任何非零值代表 TRUE。指针也是标量类型并在本章后面部分介绍。关系操作符用作值的比较并以整数值 1 或 0 的形式返回 TRUE 或 FALSE。逻辑操作符操作对应 TRUE 或 FALSE 的标量操作数并同样返回 1 或 0。

关系操作符产生 1 或 0 的 int 结果。在前面一节的例子中，计算落体落下的距离，使用了一个关系表达式来控制循环。

```
for ( int t = 1; t <= 10; t++ ) {
    // etc.
}
```

为强调控制条件，我们重写一个与上面循环等价的 while 循环。

```
int t = 1;
while ( t <= 10 ) {
    // etc.
    t++;
}
```

该循环体一直执行到条件 t<=10 的计算值不为零为止。只要 t 小于或等于 10，表达式将对 t 加 1，因而该循环是执行对应 t 从 1 到 10 的值。

关系操作符有小于<，大于>，小于等于<=和大于等于>=。还有等价符号等于==和不等于!=。这些操作符被整型、浮点以及指针操作数重载，但总生成一个 int 结果 1 或 0。

由于任何非零值都可以表示 TRUE，因而任何表达式都可以用作条件，而不仅限于那些结果为 1 或 0 的表达式。例如：

```
int t = 11;
while ( --t) {
    // etc.
}
```

循环体从 t 为 10 执行到 t 为 1。

逻辑操作符有 AND &&，OR ||，和 NOT !。一元操作符!的表达式若其操作数为非零，则其结果为零，否则为 1。例如，下式设置一个零值变量：

```
if ( !p )              // i.e. if ( p == 0 )
    p = get_a_val();
```

若两操作数皆为 0，则用||的表达式结果为 0，否则为 1。在下面的例子中，函数 error 只有当 x 的值不在 0 到 10 之间时才被调用。

```
if ( x < 0 || x > 10 )
    error ("value outside range");
```

若两操作数皆为非零，则用&&的表达式结果为 1，否则为 0。下面的式子使用了不同的逻辑表达式表示了与前一个例子相同的条件：

```
if ( !( x >= 0 && x <= 10) )
    error ("value outside range");
```

前面一个表达式可能比后面这个表达式受欢迎，因为前者容易理解。

&&和||的第二个操作数只有当需要用来确定表达式值时才被计算。若||表达式的第一个操作数为非零，则表达式结果为 1，而不管第二个操作数之值是什么。同样，若&&表达式的第一个操作数为零，而不用计算第二个操作数，表达式结果为 0。例如，在下列例子中，若 b 值为 0，则等价表达式不被计算，并且除法子表达式将被忽略。

```
if ( b && a/b == c ) {
    // etc.
}
```

&&的使用能确保发现 b 不为零，因而避免了除以 0 的可能性。

2.3 非抽象操作

C++有大量的操作符，它们允许程序员绕过类型的抽象解释而直接与它们的机器表示打交道。

整数类型在计算机上是用不同长度的位串表示的。当整型数与算术，关系或逻辑操作符一起使用时，它们的值将被抽象地解释为数值，TRUE 或 FALSE，并且其位表示细节可被程序员忽略。但有时候程序员希望直接处理这些位。

下面是源于 Peter Weinberger 的串杂凑函数。一个杂凑函数从一字符串计算一数值值并用之于确定存放位置。参数是指向计算中用来作为数值值的一字符序列的指针。变量 hash 一直被当作一个 32 位序列，直到它被用来存放其用 prime 除而得的余数时，才被当作一个数值值。

```
int
hashpjw ( char *s ) {
    const prime = 211;
    unsigned hash = 0, g;
    for ( char *p = s; *p ; p++ ) {
        hash = ( hash << 4 ) + *p;
        // assumes 32 bit int size
```

```
            if ( g = hash & 0xf0000000 ) {
                hash ^= g >> 24;
                hash ^=g;
            }
        }
        return hash % prime;
    }
```

位操作符有左移<<，右移>>，按位与&，按位异或^，按位或|和一元按位反~。

^操作符以赋值操作符^=的形式用在本例中。表达式

```
hash ^= g;
```

等价于

```
hash = hash ^ g;
```

二元操作符*，/，%，+，-，<<，>>，&，^，和|都可以用同样的方式与赋值符号连用。

sizeof 操作符给出用来表示一类型的字符个数。例如：

```
sizeof(int)
```

的结果是用来表示 int 的字节数。

```
sizeof(char)
```

的值总为 1。当其操作数是一个表达式而不是一个类型时，其结果为表达式类型的长度。该操作符可确定一个对象的动态建立所需的空间，并已结合在 C++存贮管理操作符 new 和 delete 中。有了这些操作符来协助空间分配机制，程序员将几乎不用知道类型表示的长度。

2.4 用户定义类型

用户可定义类类型以扩展基本 C++类型系统。用户还可定义类类型的操作和转换，因而可以使它们与其它类型一起使用。

下面的程序为一个含有导体，电阻和电容器的 AC 电路用公式 Z=R=jwL+1/(jwc)计算阻抗，用 V=ZI 计算电压。AC 电路的电压，电流和阻抗有两个部分，分别为一个复数的实部和虚部。在 C++中没有语言定义复数类型。该程序用一用户定义类类型来表示复数值。

```
#include "complex.h"
main() {
/*
    calculate voltage of an AC circult
*/
    const complex j( 0,1 );        // imaginary 1
    const double pi = 3.1415926535897931;
    double
        L = .03,          // inductance, in henries
        R = 5000,         // resistance, in ohms
```

```
        C = .02,          // capacitance, in farads
        freq = 60,        // frequency in hertz
        omega = 2 * pi * freq;
                          // frequency in radians/sec
    complex
        I = 12,           // current
        Z,                // impedance
        V;                // voltage
    Z = R + j * omega * L + 1/( j * omega * C );
    V = Z * I;
    V.print();
}
```

程序的输出为(60000.00, 134.13)。

头文件 complex.h 含有类类型 complex 的定义，它实现了复数的数学表示

下例使用类 complex 的一个缩略版本。

```
class complex {
    double re, im;
  public:
    complex( double r = 0, double i = 0 )
        { re = r; im = i; }
    void print();
    friend complex operator +( complex, complex );
    friend complex operator *( complex, complex );
    friend complex operator /( complex, complex );
};
```

该类定义含有成员声明以及朋友函数的声明，其对类成员进行访问。在 public 标号之后声明的成员是可被无条件存取的，而私有成员 re 和 im 只能被成员和朋友函数访问。complex 的数据成员表示隐蔽在类中的私有部分，因而该类型只有通过公开可用函数才能可用的。

名字与类相同的成员函数是一个构造器。该构造器用于建立和初始化 complex 对象，或把其它类型的值转成换类类型。该构造器以缺省参数值声明，因而它可以用零个、一个或两个参数调用，当需要时填入缺省参数。声明

```
const complex j( 0, 1);
```

是一个初始化式子，它提供了两个构造器参数j 的实部和虚部分别置为 0 和 1。声明

```
complex I = 12;
```

等价于

```
complex I ( 12, 0 );
```

缺省参数为设置 I 的虚部的参数填入 0。Z 和 V 都用缺省构造器参数初始化，因为在它们的声明中没有给出其它的初始化值。

在类 complex 中声明的其它函数被定义在类的外部：

```
void
complex::print() {
      printf("( %5.2f, %5.2f )\n", re, im );
{
complex
operator +( complex a1, complex a2 ) {
      return complex ( a1.re + a2.re, a1.im + a2.im );
}

complex
operator *( complex a1, complex a2 )
{
      return complex (a1.re * a2.re - a1.im * a2.im,
                  a1.re * a2.im + a1.im * a2.re );
}
complex
operator / ( complex a1, complex a2)
{
      double r = a2.re;    /* (r,i) */
      double i = a2.im;
      double ti;       /* (tr,ti) */
      double tr;
      tr = r < 0? -r : r;
      ti = i < 0? -i : i;
      if (tr <= ti) {
            ti = r/i;
            tr = i * (1 + ti*ti);
            r = a1.re;
            i = a1.im;
      }
      else {
            ti = -i/r;
            tr = r * ( 1 + ti*ti );
            r = -a1.im;
            i = a1.re;
      }
      return complex ( (r*ti + i)/tr, (i*ti - i)/tr );
}
```

在本例中用成员函数 print 打印 V 的结果。

```
V.print();
```

作为一个成员函数，print 可以无条件地访问类中的所有其它成员。它格式化并打印其为之调用的 complex 对象的 re 和 im 成员。

操作符函数被声明为在类 complex 内的朋友。尽管朋友函数不为类的成员，但它们象成员函数一样能访问一个 complex 对象的私有成员。操作符函数实现 complex 值的算术操作并允许在表达式中用固定符号表示 complex 操作数。

```
Z = R + j * omega * L + 1/ ( j * omega * C );
```

计算阻抗的表达式中有 int，double 和 complex 类型的操作数。当一个 int 或 double 操作数与一个 complex 操作数同时出现时，在该操作符函数被调用之前构造器会自动地把操作数转换成 complex 类型。在内部类型间的预定义转换被用来获得正确的构造器参数类型，因而该单个构造器被用作既转换 int,又和 double 操作数。当 complex 构造器被用作转换时，缺省值被当作第二个参数填入。

有了隐蔽表示和用户定义操作符以及转换，类 complex 成为一个与预定义数值类型自然地溶合的抽象数值类型。

2.5 指针和数组

指针和数组是由其它类型导出的。指针类型表示另一种类型对象的地址。它们针对数据结构的活动性跟踪动态分配的对象，并用指针算术操作符对数组元素进行访问。数组类型表示具有某一特定类型的元素系列，并常用作集合数据结构。数组的标准应用之一是串，它作为字符的系列。

指针类型在声明中用类型修饰符*以及其它类型信息进行说明。同样的符号用作指针间接引用操作符，它返回被指向的对象。间接引用的结果可用作对象的值或作为赋值表达式的左边。

```
int *p;          // p is a pointer to int
int i = 33;
p = &i;          // p set to point to i
*p = *p + 1;          // i set to 34
```

&操作符返回作为其操作数的对象之地址。地址值具有对象指针类型。

动态分配操作符 new 建立一个具有其操作数指定类型的对象并返回指向该新对象的指针。一个由 new 建立的对象可用操作符 delete 消除，delete 把一个指针作为其操作数交给一个无用对象。

```
int *p = 0;
if (!p)
     p = new int;
delete p;
p = 0;
```

上面的代码段声明了一个指针并把它初始化为 0，0 为一个特殊的空指针值并为一非法地址。检查该指针看它是被已被设置，若没有，则把一个新建立的对象地址赋给它。然后该对象被删除，并且指针被重置成空。指针的联用，作为标识的空指针的应用以及用操

此为试读,需要完整PDF请访问: www.ertongbook.com