第 1 章　走近 C/C ++

1.1　C 语言和面向过程的程序设计

1.1.1　简介

C 是 1972 年由 Dennis Richie 在贝尔实验室设计的一种通用程序设计语言。1978 年，Brian W. Kernighan 和 Dennis M. Ritchie(合称 K&R)发表了成功地用于 UNIX 的 C 编译程序。在此基础上，他们合著了影响深远的名著——The C Programming Language。他们在该书中描述的 C 语言，为后来广泛使用的 C 语言版本奠定了基础，被作为标准 C。经过多年实践，出现 C 的多种改进版本。1983 年，美国国家标准化协会(ANSI)根据这些版本制定了新的标准，称为 ANSI C。ANSI C 比原来的标准 C 有很大的改进。1987 年，ANSI 又公布了新标准——87 ANSI C。

现在流行的 C 版本有多种。Microsoft 公司、Borland 公司以及许多芯片生产厂商所用的 C 版本都略有不同。这种大同小异是不可免的，它反映 C 被广泛推广应用。现在，各种计算机都配备 C 编译器。若干年以前，人们出于无奈，不辞劳苦地用汇编语言编写许多微处理器(包括 8031 系列)的应用程序。如今，这都成为往事，几乎所有微处理器都有基于 C 的开发系统。那么，C 为何能被用户乐于采用？为什么在可预见的将来，C 还可能保其霸主地位？这是因为 C 的优点是主要的。归纳起来，C 的主要优、缺点如下。

- C 是结构化程序设计语言，它向用户提供众多的函数，利于模块化程序设计。
- C 既具有高级语言的优点，又像汇编语言那样，能直接访问物理地址，能进行位操作，能实现汇编语言的大部分功能，能直接操作硬件。所以，C 是介于高级语言和低级语言之间的一种程序设计语言。或者说，C 是一种中级语言。
- 在 C 的基础上，发展了带类的 C，即面向对象的程序设计语言 C ++ 。C ++ 的程序设计思想尽管有别于 C，但它仍然需要使用 C 的许多基本语句和设计技巧。
- C 具有较高的可移植性。C 的语句中没有依赖于硬件的输入输出语句，系统为用户提供独立的库函数来完成输入和输出操作。因此，C 本身不依赖于计算机的硬件，从而便于在硬件结构不同的机器之间实现程序的移植。
- C 提供了种类丰富的运算符和数据类型，使程序设计极为方便。
- 与汇编语言相比，高级语言生成的程序代码占用较多的存储空间，运行时间较长。

与同类高级语言(如 PASCAL、FORTRAN 等)相比,以 C 写成的源程序很简练,所生成的高效目标代码,几乎可达到汇编语言的效率。

- 语法格式灵活,给程序员提供了极大的自由空间。
- 由于某些语法规定过于灵活,有时会降低程序的易读性。
- 程序员欢迎编译器对数据类型进行严格的检查,以便在程序运行前暴露各种错误。但 C 类型检查机制相对较弱。这使得程序中的一些错误不能在编译阶段由编译器检查出来,而需要程序员在运行时仔细检查,加以排除。例如,在编译时,C 不检查数组是否越界。这类错误只在运行时才暴露。
- 运算符种类繁多,结合性和优先级等概念太复杂,初学者不易掌握。
- 与 PASCAL、FORTRAN 等高级语言一样,C 是面向过程的程序设计语言,难以编写比较复杂的程序。

1.1.2 C 源程序的组成

C 语言是一种结构化程序设计语言。C 源程序具有“模块化”结构。整个程序由顺序、选择和循环三种结构组成。在动手编程之前,要根据任务,先粗略地确定实现方案的步骤,然后继续细化每个步骤,直到可以用 C 语句来表达。总的原则是:自顶而下,逐步细化。

简单的 C 源程序只有一个源文件。这个源文件一般由三部分组成:

(1) 预处理命令;

(2) 主函数 main();

(3) 自定义函数。

ANSI C 标准允许在源程序的头部添加一些预处理命令,以改进程序设计环境,提高编程效率。系统为用户提供了许多函数库。函数是一个具有特定功能的模块。每个库有一个被称为“头文件”的接口文件。它其实是一个菜单,列出它可以提供的函数或常数。用户也可以建立自己的函数库,生成相应的头文件。用户引用库函数时,必须用预处理命令包含相应的头文件。

可以说,C 源程序由一个惟一的主函数 main()和若干个子函数组成。子函数间可以互相调用。子函数来自系统提供的或自定义的函数库,或者在源文件中被定义。整个程序的运行始于主函数,这个函数由系统调用。

1.1.3 注释

对整个程序和关键语句加上简要的注释是十分必要的。一方面,可以帮助他人理解这个程序;另一方面,也有利于程序员自己日后维护该程序。此外,在调试程序时,往往采用分块调试方法。这时,可以用注释符把不在当前调试范围的程序语句作为“注释”语句,使之不参与编译。

在 C 程序中,注释方式是这样的:在单行或多行范围内使用配对的注释符“/*”和“*/”。编译器将这两个注释符之间的字符视为程序注释,不予编译。例如:

```
/* 本函数打印 a ～ b 之间的全部素数
   返回值是素数的个数 */
⋮
a = 2; b = 100;              /* 下限为 2,上限为 100 */
⋮
```

在 C ++ 程序中,除了可以使用上述注释方法,还可以使用双斜杠“//”,在单行范围内注释。编译器遇到此注释符时,就将其后的字符视为注释,不予编译。例如,以上程序中的注释文本可改为:

```
// 本函数打印 a ～ b 之间的全部素数
// 返回值是素数的个数
   ⋮
   a = 2; b = 100;           // 下限为 2,上限为 100
   ⋮
```

1.1.4　C 程序例子

下面举例说明 C 程序的组成和基本结构。

【例 1.1】

```
/*
    这个程序打印 0 ～ 360°的 8 个等分角的三角函数,
    可选择正弦函数或余弦函数。这里使用 C 的注释方法
*/
#include <stdio.h>      /* 本程序要使用系统提供的输入/输出函数,所以要包含头文件
                           stdio.h */
#include <math.h>       /* 本程序要使用系统提供的三角函数,所以要包含头文件 math.h */
void main()             // 主函数,由系统调用
{
    int m, n;
    float d, t, Pi = 3.141593; /* Pi 是圆周率 */
    d = (2 * Pi) / 8;
    printf("打印 0 ～ 360° 的 8 个等分角的三角函数 \n");          /* 提示用户 */
    printf("请选择一个整数:0—打印正弦值, 1—打印余弦值 \n");      /* 提示用户 */
    scanf(&m);                          /* 输入整数 m 的值 */
    /* 以上几个语句组成顺序结构 */
    /* 以下根据输入的 m 值,决定打印的三角函数 */
    if(m == 0)                          // 第 1 个选择结构
        for(n = 0; n < 8; n++)          // 第 1 个循环结构
        {
            t = n * d;                  /* 顺序结构 */
            printf("%f \n", sin(t));    /* 顺序结构 */
        }
    if(m == 1)                          // 第 2 个选择结构
        for(n=0; n<8;n++)               // 第 2 个循环结构
        {
            t = n * d;                  /* 顺序结构 */
            printf("%f \n", cos(t));    /* 顺序结构 */
```

```
        }
}
```

【例 1.2】

```
/* 这个程序和例 1.1 的程序基本相同,只是在主程序中,调用自定义函数 myFunc()。由这个函
数打印三角函数。参数 m = 0 时, 打印正弦函数;m = 1 时, 打印余弦函数 */
#include <stdio.h>    /* 本程序要使用系统提供的输入输出函数,所以要包含头文件 stdio.h */
#include <math.h>     /* 本程序要使用系统提供的三角函数,所以要包含头文件 math.h */
void myFunc(int m)    /* 自定义函数,根据参数 m 的值,打印所需的三角函数 */
{
    int n;
    if(m == 0)                          // 第 1 个选择结构
        for(n = 0; n < 8; n++)          // 第 1 个循环结构
        {
            t = n * d;                  /* 顺序结构 */
            printf("%f \n", sin(t));    /* 顺序结构 */
        }
    if(m == 1)                          // 第 2 个选择结构
        for(n=0; n<8;n++)               // 第 2 个循环结构
        {
            t = n * d;                  /* 顺序结构 */
            printf("%f \n", cos(t));    /* 顺序结构 */
        }
}

void main()                             /* 主函数 */
{
    int m, n;
    float d, t, Pi = 3.141593;
    d = (2 * Pi) / 8;
    printf("打印 0 ~ 360°的 8 个等分角的三角函数 \n");              /* 提示用户 */
    printf("请选择一个整数: 0—打印正弦值, 1—打印余弦值 \n");       /* 提示用户 */
    cin << m;                           /* 输入整数 m 的值 */
    myFunc(m);                          /* 调用自定义函数 myFunc(), 打印三角函数 */
}
```

1.1.5 面向过程的程序设计特点

面向过程的程序设计(Procedure-Oriented Programming)方法诞生于 20 世纪 60 年代,其后风行全球,成为软件开发的基础。

这种程序设计方法的特点是“就事论事”,按照人们解决问题的习惯进行编程:把大问题细分为许多小任务,分而治之,各个击破。总的设计思路是:自顶向下,逐步求精。对于一个复杂过程,按其功能分解为若干个有序的基本模块。然后,再把每个模块进一步细化,直到子模块变得清晰,易于实现。每一个模块内部都可以由顺序、选择和循环等三种基本结构组成。

在面向过程的程序中,数据和处理数据的过程(函数)被分离为互相独立的实体,

即先考虑数据结构，然后据此设计处理这些数据的代码。这样，对于不同的数据结构，要编写不同的程序来完成相同的操作。另一方面，对于相同的数据结构，若操作不同，也要编写不同的程序。因此，面向过程的程序代码重用性不好。再者，这种程序设计方法总存在着这样的可能性：错误的数据调用正确的程序模块，或正确的数据调用错误的模块。

面向过程的程序设计方法虽然在处理问题的方法上符合人们思考问题的规律，但它将数据与操作数据的函数分离开来，不能如实地反映客观世界的规律。事实上，客观世界中的事物总是分门别类的。每个类有自己的数据与操作数据的方法，二者密不可分。例如，桑塔纳小汽车是小汽车类中的一类。它区别与其他类型的小汽车：它有自己的技术指标和与之配套的生产工艺。这就是所谓的“人以群分，物以类聚”。类是客观事物的一种抽象表述。每一个类是由许多具体的实体组成的。这些实体称为对象。例如，具体的某个鼠标是鼠标类的一个对象；在 Windows 应用程序的界面上，具体的命令按钮和文本框分别是命令按钮类和文本框类的对象。各类的对象之间的相互作用方式多种多样。例如，以鼠标左键单击命令按钮可以改变文本框的内容和字体，而后者又可以改变鼠标的形状，等等。如果存在多个对象，程序员就很难用面向过程的程序设计方法控制程序的规模，很难考虑诸多对象之间所有可能发生的相互作用。

在面向过程的程序设计中，通常把具有某种特定功能的模块（例如矩阵相乘、矩阵求逆等）定义成函数（或称过程、子程序），以供调用。在面向过程程序设计中，这是惟一的代码重用机制。

1.1.6　结构化程序的三种基本结构

面向过程的程序设计方法是一种结构化程序设计方法。程序由三种基本结构组成，即：

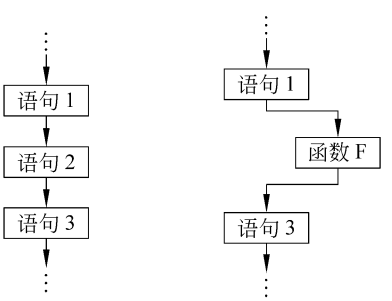
（1）顺序结构——顺序执行其中的语句；

（2）选择结构——判断某个条件是否成立，以确定程序的走向；

（3）循环结构——循环地执行某个或某几个语句，满足某个条件时，终止循环。

作为例子，图 1.1、图 1.2、图 1.3 示出以上三种结构。在实际程序中，这三种结构的出现次序视具体程序而定。

图 1.1(a)中，各个语句被依次执行。图 1.1(b)中，执行语句 1 之后，调用函数 F，其后执行语句 3。这里，函数 F 作为一个整体，可视为一个语句。

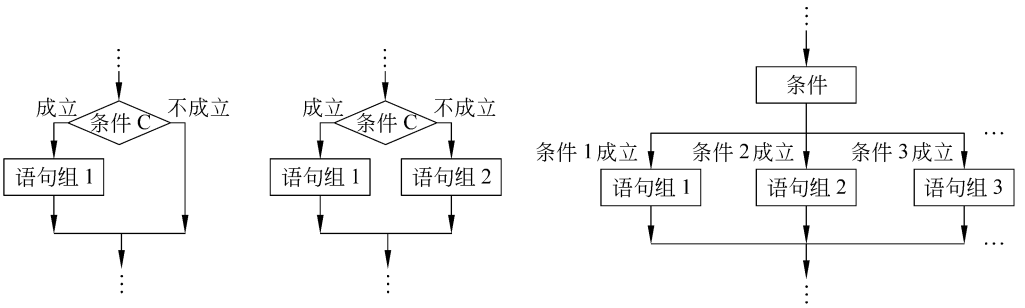


(a) 不调用函数　　(b) 调用函数

图 1.1　顺序结构

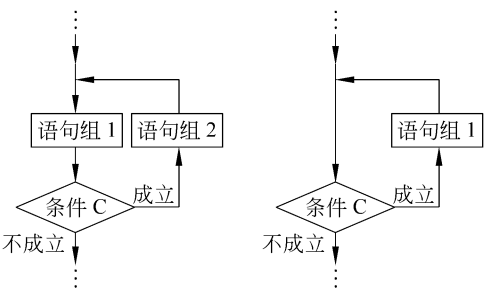
图 1.2 示出几种选择结构。这里，根据条件来判断程序的走向。

图 1.3 示出两类循环结构。程序是否循环，取决于条件 C。图 1.3(a)先执行语句组 1，然后



(a) 单分支　(b) 两分支　(c) 多分支

图 1.2　选择结构



(a) 先循环后检查　(b) 先检查后循环

图 1.3　循环结构

检查循环条件 C，如满足，就继续循环。不论条件 C 如何，语句组 1 至少被执行 1 次。图 1.3(b)则不同：先检查条件 C，如满足，就执行语句组 1；如果起始条件 C 不满足，语句组 1 连一次都不会被执行。在循环结构中，程序每循环一次，都要适当地修改某个循环变量，使条件 C 最终得以满足，从而使程序在循环一定次数后，停止循环，执行后续语句。

1.2　C++语言和面向对象的程序设计

1.2.1　C++的起源

C 语言是面向过程的程序设计语言，不适于开发大型程序。当程序规模稍大或对象之间的关系比较复杂时，程序员就很难控制程序的复杂性。例如，程序员往往需要设计带有可视界面的应用程序，即使界面上只有少数几个控件，但用 C 语言也很难设计。

为了解决上述问题，并保持 C 的简洁、高效和接近汇编语言的特点，1980 年，贝尔实验室的 Bjarne Stroutstrup 博士及其同事开始对 C 进行改进和扩充。这种新型的 C 最初称为“带类的 C”；1983 年，改称 C++。其后，经过不断完善和发展，才演变为今日的 C++。

C++将 C 作为它的一个子集。C++原意是 C plus plus，表示 C++继承和发展了 C。

本书所讲的编程语句，如未特别声明，均适用于 C 和 C ++ 。

C ++ 作为面向对象的程序设计(Object-Oriented Programming，OOP)语言，对 C 的增强表现在以下两个方面。

(1) 在原来面向过程的基础上，添加了许多新功能。例如：函数重载，用运算符 new 和 delete 分别申请和删除动态内存，使用新的 I/O 流库，等等。

(2) 增加了面向对象的机制。许多 C 程序不经任何修改，便可以在 C ++ 环境中运行。

面向对象与面向过程是两种不同的程序设计思路。读者学完本书后，就会悟出面向对象程序设计方法的优越性。虽然由于兼容性，在 C ++ 环境也可以用 C 语言按照面向过程的程序设计思路编程，但这岂不是“穿新鞋走老路”吗？

1.2.2　C ++ 的兼容性

使用 C ++ 前，必须安装 C ++ 编译系统。在 Microsoft VC ++ 6.0 开发环境中，C ++ 和 C 完全兼容。例 1.3 的程序在 Microsoft VC ++ 6.0 开发环境中能通过编译。在这个程序中，兼用 C ++ 独有的头文件、标准输出流和注解方法。

【例 1.3】

```
#include <stdio.h>                    /* C 和 C++ 都有这个头文件 */
#include <iostream.h>                 // C++ 独有的头文件
{
    int a = 10;
    int b = 20;
    printf("a = %d \n",a);            /* C 和 C++ 都有的输出函数 */
    cout <<" b =" << b << endl;       // C++ 独有的标准输出流
}
```

1.2.3　面向对象的程序设计语言——C ++ 的特点

C ++ 改进和扩充了 C 的类型系统。最重要的扩充是支持面向对象的程序设计。现在，“面向对象”一词非常时髦。许多可视化程序设计语言，如 Microsoft VC ++ ，被标榜为面向对象的程序设计语言。那么，什么是面向对象的程序设计技术？它有哪些特征？对此，本小节将力图讲解清楚。

C ++ 的创始人对自然界和人类社会观察入微，才把原来面向过程的 C 语言扩充为面向对象的 C ++ 。

面向对象的程序设计语言有什么特征？

简单说来，面向对象程序设计方法如实地反映了客观事物的存在规律，将数据和操作数据的方法(函数)视为一体，作为一个互相依存、不可分割的实体来处理。这就是“物以类聚，人以群分”。面向对象程序设计语言有以下三个特征。

- 封装性：类将数据和操作封装为用户自定义的抽象数据类型。
- 继承性：类能被复用，具有继承(派生)机制。
- 多态性：类具有动态联编机制。

仔细观察自然界和人类社会中的事物，就容易理解这三个特征。

(1) 封装性(Encapsulation)

封装性的第一层意思就是数据抽象(Data Abstraction)。

我们可以自定义数据类型，这种数据类型称为类。类中包含数据成员，还包含操作这些数据的成员函数。例如，可以把复数的实部和虚部作为数据成员，连同操作复数的成员函数(计算两复数之和、差、乘积和商，以及计算复数的幅值和幅角等)封装为一个称为complex 的类。桑塔纳小汽车也可以自成一类，其数据成员有几何尺寸、发动机型号、耗油量、最大车速等，其成员函数有加工工艺、驾驶方法、维修方法等。而发动机又可以自成一类，具有自己的数据成员和成员函数。

类是一种数据抽象，所说明的是许多实体的共性。实体则是类的对象。例如，整数自成一类，被声明的某个变量则是它的对象。桑塔纳小汽车是诸多小汽车中的一类，但领有牌照而奔驰于路上的桑塔纳小汽车则是该类的一个对象。

我们既可以按照客观世界已明显存在的事物进行分类，定义抽象数据类型，还可以根据编程的需要，自定义抽象数据类型。例如，我们可以将数字滤波器的参数和复数类数据，连同滤波器的运算函数封装为一个称为 filter 的类。

其实，C 语言本身就为用户封装了许多类，例如，整数类和单精度类的数据成员分别是整数和单精度数，各有不同的运算函数。C 语言虽然允许用户定义结构体，但结构体内只有数据成员，而不允许有成员函数。

在 C ++ 中，用户可以把数据和处理数据的函数封装成一个类，其意义不可低估。类中有数据和处理数据的函数，就相当于一个小计算机。

封装性的第二层意思就是信息隐蔽(Information Hiding)。

在一定范围内取值的自然数被 C/C ++ 系统封装为整数类，类中有操作整数的函数。系统只告诉用户怎样操作整数，而隐蔽整数类中的函数。这就是信息隐蔽，事实上，用户没有必要过问其中的实现细节。

在现实生活中，信息隐蔽不乏其例。昔日的电子管黑白电视机使用暴露于外的分立元件，如果需要维修，就往往要遍查机内各点，不胜其烦。时至今日，彩色电视机的功能更多，但其构成却相对比较简单，因为它用的是大规模集成电路芯片。厂家不会把机内各种芯片和整机的组成等技术细节告诉用户，这就是信息隐蔽，而用户也完全没有必要探究这些细节。事情是非常简单的：厂家只须提供手册，说明电视机的技术指标和使用方法，而用户只要正确使用即可。若有故障，用户可通知厂家上门维修。所以，时代进步了，人们才能省心省事。

对于 C ++ 的类来说，情况也是如此。类有一个清晰的接口。这个接口是一个菜单，向用户宣示类中有什么数据成员，有什么公有成员函数可供调用。数据成员一般是私有的，只有通过公有的成员函数才能访问。这样，数据成员就受到保护。公有成员函数可以用来实现类对象之间的沟通，其实现细节可以隐藏在源文件中，不向用户提供。向用户提

供的只是这个源文件编译所得的目标代码。这种信息隐蔽技术能够保护数据成员和实现代码，使用户无法私自改变，无法获知诸多实现细节。事实上，开发者向用户提供了可靠的类（例如矩阵类和多项式类等），用户完全没有必要追究个中细节，他（她）只须全力以赴，开发自己的应用程序即可。由此看来，信息隐蔽是好事，它体现了技术进步。

（2）继承性（Inheritance）

在客观世界中，从横的方面看，诸多事物分门别类；从纵的方面看，事物有继承性。被继承的类称为基类，由基类派生出来的类称为派生类。无选择的继承可能导致退化。继承应是为了推陈出新。因此，完备的继承机制应能实现：

- 完全继承基类的优点；
- 改进基类的缺陷；
- 添加新的特点。

例如，A 型小汽车可归为一类。这类小汽车（具体对象）都具有某些属性，即数据成员（例如车的尺寸、颜色、最大车速等）。同时，它们还有某些加工工艺，即成员函数（例如喷漆工艺、各种装配工艺等）。当开发新的 B 型小汽车时，可以在 A 型的基础上进行研制：保留原有的喷漆工艺，改进焊接技术，增加音响设备等。

通过继承和组合（引用新的部件），可以开发出花样翻新的产品。

与此相仿，C ++ 中的类有完备的继承机制。在程序设计时，先设计具有一般性能的基类，然后由此派生新类，有选择地继承基类的数据成员和成员函数，并作修改和增补。这是一个以类为基础的细化过程。这个细化过程与面向过程的程序设计细化过程是不同的。在 C ++ 中，仍然可以通过设计函数来实现代码重用，但代码重用主要是通过继承机制来实现的。例如，矩阵类 Matrix 定义了矩阵的数据成员（行、列数和数据指针）和实现矩阵各种运算的成员函数。其后，在开发数字滤波器时，可以定义 Filter 类继承 Matrix 类，使前者能继承后者的成员函数的代码。

（3）多态性（Polymorphism）

客观事物有多态性。多态性与事物的类有关。Movement 一词可以指人们躯体的活动，也可以指政治、社会或思想上的活动。Move 一词可以理解为“搬动”，也可以理解为“搬家”。这种一词多义现象是一种多态性。每一个词的确切意义与上下文（语境）涉及的对象有关，这也许就是人类语言魅力之所在。我们注意到上述多态性与类的继承无关，这种多态性可称为静态多态性。另外还有一种与继承有关的多态性，称为动态多态性。例如，War 指的是一类战争行为，但所指的具体形式则与历史时代有关。“三国演义”中的战争指的是金戈铁马；近代的战争指的是机械化战争；而当今的战争则演进为无接触的信息化战争。

C 语言不支持任何多态性。C 程序不允许同名函数存在。程序中，函数的调用必须由程序员预先设定。随着程序规模的变大，程序员就难以控制函数的调用了。

C ++ 支持静态多态性，它允许多个函数使用同一个名字，编译时，系统根据函数参数的个数、类型和次序来决定调用哪个函数。C ++ 也支持动态多态性。这种多态性由继承机制支持。建立了类层次之后，各个类可以有同名函数。编译时，不能确定调用哪个函

数。程序运行时，根据对象所在的类，系统就确定被调用的函数。由于C++支持两种多态性，所以，当程序规模变大时，程序员不必操心怎样正确调用函数，这一切都将由系统办妥。

1.2.4 面向对象的程序设计特点

从总体上看，面向过程程序设计方法和面向对象程序设计方法的思路是很不相同的。

1.1.5小节已经介绍了面向过程的程序设计特点。面向过程程序设计是围绕功能进行的，程序所用的数据和操作这些数据的程序代码是分开的。设计时，程序员先定义一些数据，然后把大任务细分为许多小功能模块，每个功能由一个函数来完成。各个函数是在完成程序功能的过程中被依次调用的。所以，面向过程的程序本质上是过程驱动的。

此外，在面向过程的程序中，所有数据是公开的。一个函数可以使用和改变任意一组数据，而一组数据又可能被多个函数使用。这样，在程序规模较大，数据很多，操作种类繁多时，程序员就往往顾此失彼，难以跟踪数据的变化，从而极易出错。进一步说，即使对任务做小小的修改，也往往要对程序大检查、大修改。

面向对象的程序设计思路则不同。程序员根据具体情况，先设计一些类。每个类有数据成员和操作这些数据的成员函数。然后，定义各个类的对象，并将数据赋给各个对象。对象的数据是私有的，外界只能通过公有的成员函数才能访问该对象的数据。这样就很好地保证数据能被正确使用，大大地减少了出错的可能性，而且程序员也易于对数据进行跟踪。

每个对象是数据和操作代码的完整结合体，这无异于一个独立的小计算机。各个对象通过消息传递而相互作用。所以，面向对象的程序本质上是事件驱动的。这一点很重要，它使得一个原先很复杂的程序变得简单清晰。这种优势在可视化程序设计中极为明显。图1.4所示的简单例子最能说明问题。

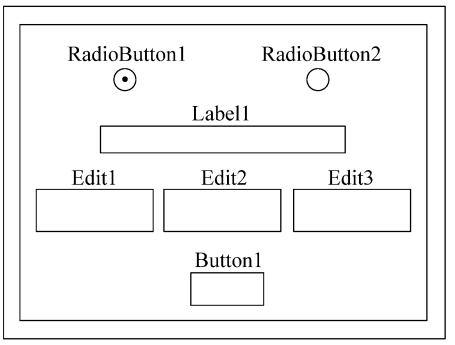


图1.4 一个Windows应用程序的界面

图1.4示出一个Windows应用程序的界面。所谓界面就是屏幕上显示的一个窗体，其上放置一些人机对话控件。图1.4右侧标出了各个控件的名称、类别和功能。这个应用程序很简单。用户在文本框Edit1和Edit2中分别输入第1、第2操作数，选中单选按钮RadioButton1时，进行乘法运算。选中RadioButton2时，进行除法运算。当除数不为