ISBN 978-7-302-18078-4

21世纪高等学校计算机专业实用规划教材

C++程序设计实用教程

李青 周美莲 编著

清华大学出版社
北京

目　录

源代码目录

第 4 章　变 量 设 计

在本章中，通过几种不同类型的算法实现着重讨论算法实现中的变量设计问题，包括函数的参数、局部变量、标志变量的设计与应用。从本章起，将围绕所谓的“评委评分”程序的设计和优化、程序功能的扩充逐步展开讨论。本章涉及一些基本算法，这些算法的设计思想是朴素的。

变量设计是程序设计的重要环节，初学者应该重视这一环节。熟练后，这一环节将成为程序员程序设计时的一种“自觉行为”。

4.1　穷举计算

通过一个具体实例，介绍“实际问题 → 数学模型 → 计算方法 → C++程序 → 计算机运行程序”这一计算机求解问题的全过程。本节重点介绍穷举法程序设计。

4.1.1　“百钱买百鸡”问题

例 4.1　【“百钱买百鸡”问题】公元前 5 世纪，我国古代数学家张丘建在《算经》一书中提出了“百鸡问题”：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。百钱买百鸡，问鸡翁、母、雏各几何？

【数学模型】记 x, y, z 分别表示购买鸡翁，鸡母，鸡雏的数量（$x, y, z \in \mathbb{Z}$），则有

$$\begin{cases} x + y + z = 100 \\ 5x + 3y + z/3 = 100 \end{cases}$$

上述方程组有三个未知数仅有两个方程，是一个不定方程。它在非负整数范围内可能有多组解。求解不定方程的数学方法需要较多的理论知识，存在较大的困难。

【算法设计】显然有 $0 \leqslant x \leqslant 100/5,\ 0 \leqslant y \leqslant 100/3$，且 $x, y \in \mathbb{Z}$。即

$$\begin{cases} x = 0, 1, 2, \cdots, 20 \\ y = 0, 1, 2, \cdots, 33 \\ z = 100 - x - y \end{cases}$$

亦即，该问题可以被限制在如上所述的一个较小范围——**包含解集的某有限集合**内讨论。利用计算机快速运算的能力，在该有限集合中逐个进行搜索（试算），以找出该问题的所有解。这种处理（算法创作）策略被称为**穷举法**或**枚举法**。这种以简御繁的策略是计算机解题的一种常用方法。**穷举法将求解问题质的困难转化成量的重复**。但是，当可能的解集中元素个数巨大时，该方法失效。尽量裁减搜索范围是穷举法成败的关键。

本问题可能的解集是三维空间中的点，由于有约束条件 $x+y+z=100$，可将解集投影到任意一个二维坐标平面上。例如，将解集投影到xoy平面上，得到的点应该被包含在集合 $A = \{(x, y) | x = 0, 1, 2, \cdots, 20; y = 0, 1, 2, \cdots, 33\}$ 中，该集合共有 $21 \times 34 = 714$

个元素。因此，仅需要对这 714 种情形逐个试算，判断是否满足题意，并将满足者输出即可。即需要对 (x, y) 每一对组合**反复**地进行**判断**操作（必要时进行输出）。

值得指出的是，若考察 (y, z) 取值的各种组合，将有 $34 \times 101 = 3434$ 种情况。显然此时的穷举操作计算量将近是考察 (x, y) 组合的 5 倍。由于有约束条件，更不必去讨论 (x, y, z) 各自独立取值的 $21 \times 34 \times 101 = 72114$ 种组合，而且还要另外判断 $x + y + z$ 是否等于100，其计算量是考察 (x, y) 组合的100多倍。这说明在穷举计算中，搜索范围的不同裁减可导致计算效率的巨大差别。

【计算流程】将集合 A 的 714 个元素描在平面坐标系中，将形成一个矩形点阵。遍历这 714 个元素可按“行”或按“列”依次扫描。例如，按“列”的具体扫描过程如下：

① 确定购买鸡翁的数量 x（从 0 起，逐次增加 1，直到 20 为止）。

② 对于**每一个**确定的 x 而言，再确定购买鸡母的数量 y（从 0 起，逐次增加 1，直到 33 为止）。

③ 对于每一个确定的 (x, y) 组合，计算购买鸡雏的数量 $z = 100 - x - y$；计算所需要的金额 $s = 5x + 3y + z/3$；判断 s 是否恰好为 100，若是，则对应的 (x, y, z) 便是该问题的一组解（输出该组解）。处理下一个组合。

显然，上述的三个过程是逐层嵌套的。按照自顶向下、逐步求精的设计方法分解成图 4-1 所示的计算流程图。

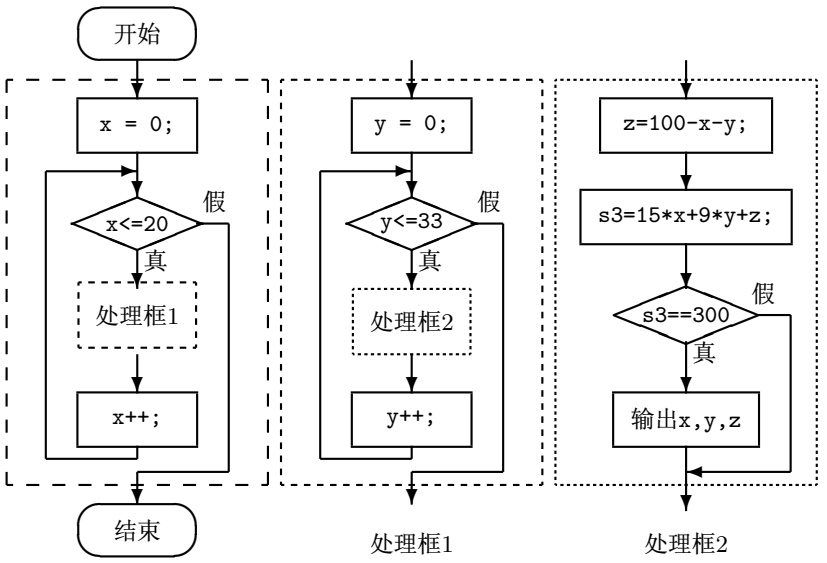

图 4-1 “百钱买百鸡”问题计算流程图

在左侧的总框图中，留有一个待进一步细化的部分（处理框 1）；同样，在处理框 1 中，也留有一个待细化的部分（处理框 2）。它们是按层次完全嵌套的，从上一层看待下一层时，处理框是一个整体。逐层细化的部分见图 4-1 中用不同虚矩形框表示的部分。每一个虚矩形框内部均又由若干语句组成，从宏观来看，虚矩形框是“单入口、单出口”的，能保证该处理框完全嵌入到其上一层框图之中。

【C++ 程序】根据上述计算流程图，按照 C++ 语言提供的表达方式，采用合适的循环结构、选择结构等 C++ 语句，写成如下 C++ 源程序。

源代码 4.1 百钱买百鸡问题

```
// HundredChicken.cpp    源程序文件名
#include <iostream>
using namespace std;

int chicken100();              // 函数原型用于函数声明

int main()
{
    chicken100();              // 主函数起调度作用，尽可能简单
    return 0;
}

int chicken100()
{
    int cocks, hens, chicks, n=0;
    for(cocks=0; cocks<=20; cocks++)    // 可能购买公鸡的数量 0～20
        for(hens=0; hens<=33; hens++)   // 可能购买母鸡的数量 0～33
        {
            chicks = 100 - cocks - hens;  // 购买小鸡的数量
            if(3*100 == 3*(5*cocks + 3*hens) + chicks)
            {                                  // 找到一个解
                cout << "鸡翁␣" << cocks
                     << "␣只，鸡母␣" << hens
                     << "␣只，鸡雏␣" << chicks
                     << "␣只。" << endl;  // 输出解
                n++;                          // 解集元素个数增 1
            }
        }
    cout << "共有␣" << n << "␣个解。" << endl;
    return n;
}
```

【运行程序】程序经过编译、连接后生成可执行文件，运行可执行文件的结果如下。

```
鸡翁 0 只，鸡母 25 只，鸡雏 75 只。
鸡翁 4 只，鸡母 18 只，鸡雏 78 只。
鸡翁 8 只，鸡母 11 只，鸡雏 81 只。
鸡翁 12 只，鸡母 4 只，鸡雏 84 只。
共有 4 个解。
```

将总框图中的虚矩形框写成函数 chicken100 由主函数调用，其主要部分是两重循环及一个分支选择语句的嵌套，与流程图（如图 4-1 所示)对应。整个循环结束时 cocks 的值为 21（即继续循环条件 cocks <= 20 不成立时）；此时 hens 的值为 34（这已经是

该变量第 21 次，从 0 逐步递增到 34，总共被改变 714 次）；此时 chicks 的值应该是 100－20－33=47。

值得指出的是，若将第 20 行中的条件判断改成 100 == 5*cocks + 3*hens + chicks/3，则程序的运行结果将给出 7 组解，其中有三组是错误的。若改成100 == 5*cocks + 3*hens + chicks/3.0，则程序运行正确。但是，这有赖于两个近似表示的浮点数之间的精确比较，这种做法是应该尽量避免的。本例中，将比较的双方均扩大到三倍成为整数后进行精确的比较。

另外，将常量写在关系运算符“==”的左侧有利于减少将关系运算符误用成赋值运算符“=”的机会，因为编译器会指出“if(3=a)”中的语法错，而“if(a=3)”却能够通过编译。由于赋值表达式“a=3”的值为 3（非零，即 true），上述判断相当于“if(true)”，这不是程序员的本意。

4.1.2 判定素数

例 4.2 给定一个正整数 n，判断其是否为素数（素数亦称为质数，即只有 1 及自身两个平凡因数的正整数。正整数 1 不是素数）。

【算法设计】为了判断给定的正整数 n 是否有非平凡的因数，简单的方法是穷举搜索。因为 n 的非平凡因数一定不超过 n。故可从 2 起到 n-1 逐个进行判断。这样处理的计算量大约为 n。进一步分析以裁减搜索范围。若 n 有一个非平凡因数 a，则 n/a 亦为 n 的一个非平凡因数，并且其中必有一个因数不超过 $\sqrt{n}$。从而，实际计算只需要从 2 起穷举判断到 $\lfloor\sqrt{n}\rfloor$（向下取整）即可。这样处理的计算量约为前一种处理方法的 $1/\sqrt{n}$ 倍，节省的计算量十分可观！

源代码 4.2 判定素数

```
1  // isprime.cpp  判定素数
2  #include <iostream>
3  #include <iomanip>
4  #include <cmath>
5  using namespace std;
6
7  int isprime(unsigned int n)
8  {
9      if(n<2) return 0;
10     unsigned int k, m;
11     m = (unsigned int)sqrt(double(n));  // 数据类型强制转换
12     for(k=2; k<=m; k++)
13         if(n%k==0)
14             return 0;
15     return 1;
16 }
17 int main()
18 {
19     const unsigned int N = 1000;
20     unsigned int n, m=0;
```

```
21      for(n=2; n<=N; n++)
22          if(isprime(n))
23          {
24              cout << setw(4) << n;
25              m++;
26          }
27      cout << "\n\n" << N << "␣以内共有素数␣"
28           << m << "␣个。" << endl;
29      return 0;
30  }
```

程序中有如下两种情形结束第 12～14 行循环语句。

① 在该循环体内有机会执行到 return 语句，表明此时的 k 是 n 的一个非平凡因数，故 n 是合数（不是素数），此时已无必要判断到底，可立即返回结果。

② 继续循环的条件不成立，此时 k==m+1成立。可见，变量 k 除了计数、作为除数外，它还可充当标志。

在用循环语句实现穷举计算，并且有提前结束循环语句情形时，循环控制变量常常也用作标志。在循环语句结束后，通过判断循环控制变量的值可知该循环语句的实际执行情况。上述函数可以改写成如下形式。

```
1  #include <cmath>
2  int isprime(unsigned int n)
3  {
4      if(n<2) return 0;
5      unsigned int k, m;
6      m = (unsigned int)sqrt(double(n));
7      for(k=2; k<=m; k++)
8          if(n%k==0) break;    // 提前结束循环，但不立即退出函数
9      if(k<=m)                 // 循环语句结束后根据 k 的值进行判断
10         return 0;
11     else
12         return 1;
13 }
```

程序中的辅助变量 m 具有重要的作用。若去掉该变量而将循环语句的首部写成 for(k=2; k<=sqrt(double(n)); k++) 则执行了许多重复计算。当然，可以写成 for(k=(unsigned int)sqrt(double(n)); k>=2; k--)，而执行一次 sqrt(double(n)) 的计算。

关于 for 语句中定义的变量的生命期，标准 C++ 规定 for 语句中定义的变量的生命期及作用域仅限于其所在的循环语句。例如：

```
1  for(int i=1; i<10; i++)
2  {
3      for(int j=1; j<10; j++)
4          cout << i << "*" << j << "=" << setw(2) << i*j;
5      cout << endl;        // 不能访问 j
```

```
6 }
7 // 不能访问 i 及 j
```

VC++ 编译器将这种变量的生命期和作用域延长到其所在函数返回为止。这是非标准的。如下函数:

```
1 #include <cmath>
2 int isprime(unsigned int n)
3 {
4     if(n<2) return 0;
5     unsigned int m;
6     m = (unsigned int)sqrt(n);
7     for(int k=2; k<=m; k++)
8         if(n%k==0) break;    // 提前结束循环，但不立即退出函数
9     if(k<=m)    // 循环语句结束后，仍可访问 k。根据 k 的值进行判断
10        return 0;
11    else
12        return 1;
13 }
```

仅能在 VC++ 中使用。在 GCC 编译器中认为第 9 行中变量 k 未定义而错。而语句

```
1 for(int i=0; i<10; i++)
2     cout << i << '␣';
3 cout << endl;
4 for(int i=0; i<10; i++)
5     cout << char('A'+i) << '␣';
```

符合 C++ 语言标准，能在 GCC 编译器通过，却被 VC++ 编译器认为第 4 行变量 i 重复定义而错。遇此情形，可将变量 i 定义为所在函数的局部自动变量，回避 for 循环语句中定义变量的生命期因编译器不同而异的问题。例如:

```
1 int main()
2 {
3     int i;
4     for(i=0; i<10; i++)
5         cout << i << '␣';
6     cout << endl;
7     for(i=0; i<10; i++)
8         cout << char('A'+i) << '␣';
9     return 0;
10 }
```

4.2 迭代计算

迭代是一种常用的有效算法。本节介绍几个典型迭代计算的程序实现。

4.2.1 牛顿迭代法

在第 1 章中提到计算非负实数算术平方根（ $\sqrt{d}$ ）的牛顿迭代法[1]：

$$\begin{cases} x_0 = 1 \\ x_n = x_{n-1} - \left(x_{n-1} - \dfrac{d}{x_{n-1}} \right) /2 \quad n = 1, 2, 3, \cdots \end{cases}$$

为了与 cmath 中声明的标准函数区别，将函数原型设计成如下形式：

```
double mysqrt(double);      // 形参类型是重要的，形参名称可缺省
```

函数定义如下：

源代码 4.3 自定义的平方根函数

```
1  double mysqrt(double x)
2  {
3      double delta, y=1, epsilon=1e-8;
4
5      do
6      {
7          delta = (x/y - y)/2;
8          y += delta;
9      }while(delta>=epsilon || delta<=-epsilon);
10     return y;
11 }
```

4.2.2 级数计算

1. 指数函数

由高等数学的无穷级数理论可知，一些常用的数学函数可表示成无穷级数的和。通过计算无穷级数的前若干项部分和达到近似计算函数值的目的。本书不讨论级数的收敛性、收敛速度等数学问题，而是直接应用数学结论进行程序实现。

例 4.3 根据如下级数计算指数函数的值。

$$\mathrm{e}^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} + \cdots \qquad (-\infty < x < \infty)$$

[1]解方程 $f(x) = 0$ 的牛顿迭代公式为

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})} \quad n = 1, 2, 3, \cdots$$

任取一满足 $f'(x_0) \neq 0$ 的 x_0 作为迭代计算的初值。对于预先给定的适当小的正数，如 $\varepsilon = 10^{-8}$，若前后两次迭代计算的差别$|x_n - x_{n-1}| < \varepsilon$，则可将$x_n$作为方程的近似解。考虑方程 $f(x) = x^2 - d = 0$ $(d \geqslant 0)$，有导函数$f'(x) = 2x$，即得到上述计算非负实数算术平方根的迭代公式。

【分析】在计算机上计算无穷级数的部分和，到底需要计算多少项呢？一般采用精度控制法。取一个适当小的正数，例如，$\varepsilon = 10^{-8}$，当前后两次部分和的差（即某一项）的绝对值小于 ε 时便截断，并以此部分和作为级数的近似值。

级数求和计算是一个重复过程。一般地，根据级数项的递推公式依次计算出各项，再将其累加到部分和中。上述数学公式经过变形，可写成下面的迭代（递推）公式：

$$\begin{cases} s_0 = 1, p_0 = 1 \\ p_n = p_{n-1} \times x/n & n = 1, 2, 3, \cdots \\ s_n = s_{n-1} + p_n \end{cases}$$

这种递推公式中涉及的运算是不变的，只有数据在发生变化，这就是所谓的循环不变式。

由于我们并不关心计算的中间结果，可将逐步计算出的项 （p_n） 及部分和 （s_n） “原地覆盖存储”：用计算得到的新值 （p_n） 及 （s_n） 分别覆盖前一步的旧值 （p_{n-1}） 及 （s_{n-1}）。因此，只需要少量的存储单元便能实现上述计算。也就是说，将上述数学迭代公式中的下标去掉，就是算法的伪代码。

```
步1  epsilon=1e-8;
     s = p = n = 1;
步2  若 fabs(p) >= epsilon 则循环
          p *= x/n;
          s += p;
          n++;
步3  s 即为所求。
```

源代码 4.4　自定义的指数函数

```
1  double myexp(double x)
2  {
3      int n;
4      double s, p, epsilon=1e-8;
5
6      s = p = n = 1;
7      do
8      {
9          p *= x/n;
10         s += p;
11         n++;
12     }while(p>=epsilon || p<=-epsilon);
13
14     return s;
15 }
```

特别值得注意的是，计算第 n 项时，**切勿分别计算 x^n 及 $n!$，然后计算它们的商**。因为它们均可能是值非常大的数，可能出现数据溢出或出现较大的误差（参见练习 4 的 2（2））；再做除法，又可能出现更大的误差。

2. 正弦函数

例 4.4 计算正弦函数的值。正弦函数的级数展开式为交错级数：

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!} + \cdots \qquad (-\infty < x < \infty)$$

【算法描述】

步1 epsilon=1e-8;
s = p = x;
n = 3;
步2 若 fabs(p) >= epsilon 则循环
p *= -x*x/(n*(n-1));
s += p;
n +=2;
步3 s 即为所求。

源代码 4.5 自定义的正弦函数

```
double mysin(double x)
{
    int n;
    double s, p, epsilon=1e-8;

    s = p = x;
    n = 3;
    do
    {
        p *= -x*x/(n*(n-1));
        s += p;
        n += 2;
    }while(p>=epsilon || p<=-epsilon);

    return s;
}
```

【综合测试】采用多文件结构测试上述自定义的数学函数。mymath.cpp，mymath.h 为一对文件（可将其移植到其他程序中）。将上述三个自定义函数录入到 mymath.cpp 文件中（略）；头文件 mymath.h 中的内容为函数原型，作用是函数声明。

```
// mymath.h
#ifndef MYMATH_H
#define MYMATH_H

double mysqrt(double);
double myexp(double);
double mysin(double);

#endif
```

测试函数及主函数编辑到另一个源程序文件 MyMathTest.cpp 中。

源代码 4.6 自定义数学函数综合测试程序

```
// MyMathTest.cpp
#include <iostream>
#include <cmath>          // 用标准函数与之比较
#include "mymath.h"
using namespace std;

int mysqrtTest()
{
    double x, err, max_err=0;
    for(x=0; x<=100; x++)
    {
        err = mysqrt(x) - sqrt(x);     // 自定义函数与标准函数之差
        if(fabs(err) > max_err)
            max_err = err;             // 找出误差绝对值的最大值
    }
    cout << "mysqrt(x)-sqrt(x),␣(␣␣0␣≤␣x␣≤␣100),␣␣Max_err␣:␣"
         << max_err << endl;
    return 0;
}
int myexpTest()
{
    double x, err, max_err=0;
    for(x=-10; x<=10; x++)
    {
        err = myexp(x) - exp(x);
        if(fabs(err) > max_err)
            max_err = err;
    }
    cout << "myexp(x)␣-␣exp(x),␣(-10␣≤␣x␣≤␣␣10),␣␣Max_err␣:␣"
         << max_err << endl;
    return 0;
}
int mysinTest()
{
    const double PI=3.1416;
    double x, err, max_err=0;
    for(x=-2*PI; x<=2*PI; x+=0.1)
    {
        err = mysin(x) - sin(x);
        if(fabs(err) > max_err)
            max_err = err;
    }
    cout << "mysin(x)␣-␣sin(x),(-2␣π≤␣x␣≤␣2␣π␣),␣Max_err␣:␣"
         << max_err << endl;
    return 0;
}

```

```
48 int main()
49 {
50     mysqrtTest();
51     myexpTest();
52     mysinTest();
53     return 0;
54 }
```

程序的运行结果：

```
mysqrt(x)-sqrt(x), (  0 ⩽ x ⩽ 100),  Max_err : 7.45058e-009
myexp(x) - exp(x), (-10 ⩽ x ⩽  10),  Max_err : -9.22673e-010
mysin(x) - sin(x),(-2 π⩽ x ⩽ 2 π), Max_err : 2.83173e-010
```

从结果可见，自定义的几个数学函数与标准数学函数的计算结果误差不超过所取的精度控制 $\varepsilon = 10^{-8}$。

4.2.3 最大公因数和最小公倍数

例 4.5 求两个整数 m, n 的最大公因数 (m, n) 和最小公倍数 $[m, n]$。

【分析】首先有 $(m, n) = (|m|, |n|)$ 及 $[m, n] = [|m|, |n|]$；其次考虑非负整数 m, n

$$(m,n) = \begin{cases} m & n = 0, m \neq 0 \\ \text{不存在} & n = 0, m = 0 \\ (n, m\%n) & n \neq 0 \end{cases}$$

$$[m,n] = \begin{cases} \dfrac{m \times n}{(m,n)} & n \neq 0, m \neq 0 \\ \text{不存在} & n = 0 \text{ 或 } m = 0 \end{cases}$$

最后仅需讨论 m, n 全为正整数的情形。由带余除法：

$$m = q_1 n + r_1 \qquad 0 \leqslant r_1 < n$$

可知，m, n 的公因数集合与 n, r_1 的公因数集合相同，因而其最大公因数相等。同理，由

$$n = q_2 r_1 + r_2 \qquad 0 \leqslant r_2 < r_1$$

可知，$(m,\ n) = (n,\ r_1) = (r_1,\ r_2)$。反复这个过程（称为辗转相除法），由于 $0 \leqslant \cdots < r_2 < r_1 < n$，必定有某 $r_k = 0$，则 r_{k-1} 即为所求。

根据上述讨论，计算两个非负整数的最大公因数的函数设计如下。

源代码 4.7 计算最大公因数

```
1 // 最大公因数 Greatest Common Divisor
2 unsigned int gcd(unsigned int m, unsigned int n)
3 {
4     unsigned int r;
5     while(n)
```

```
6       {
7           r = m%n;
8           m = n;
9           n = r;
10      }
11      return m;
12  }
```

m, n 的最小公倍数存在时其函数可如下定义。

源代码 4.8 计算最小公倍数

```
1  // 最小公倍数 Lowest Common Multiple
2  unsigned int lcm(unsigned int m, unsigned int n)
3  {
4      unsigned int gcd(unsigned int, unsigned int);
5                   // 函数原型用于函数声明，声明 gcd 函数为局部函数
6      unsigned int g = gcd(m, n);
7      if(g==0)
8          return 0;
9      else
10         return m/g*n;    // 显然 m%g 为 0，尽量避免可能由 m*n 造成的溢出
11 }
```

例如：

$m = 36, n = 14$ 时，

$$36 = 2 \times 14 + 8$$
$$14 = 1 \times 8 + 6$$
$$8 = 1 \times 6 + 2$$
$$6 = 3 \times 2 + 0$$

亦即 $(36, 14) = (14, 8) = (8, 6) = (6, 2) = (2, 0) = 2$。得到 2 为 36 与 14 的最大公因数。36 与 14 的最小公倍数为 $36/2 \times 14 = 252$。

4.3 标志变量的设计与应用

4.3.1 整除问题

例 4.6 编程实现输入一个整数，判断其是否能被 3、5 或 7 整除，并输出以下信息之一。

（1）能同时被 3, 5, 7 整除。

（2）能被两个数（要指出哪两个数）整除。

（3）仅能被一个数（要指出哪一个数）整除。

（4）不能被 3, 5, 7 整除。

【分析】判断整数 n 能否被 3 整除的方法是判断 n 除以 3 的余数 n%3 是否为 0。方法如下。

【方法一】

源代码 4.9 整除问题

```
// 整除问题1.cpp
#include <iostream>
using namespace std;
void f(int n)
{
    if(n%(3*5*7)==0)
        cout << n << "␣能同时被␣3,␣5,␣7␣整除。";
    else if(n%3==0 && n%5!=0 && n%7!=0)
        cout << n << "␣仅能被一个数␣(3)␣整除。";
    else if(n%3!=0 && n%5==0 && n%7!=0)
        cout << n << "␣仅能被一个数␣(5)␣整除。";
    else if(n%3!=0 && n%5!=0 && n%7==0)
        cout << n << "␣仅能被一个数␣(7)␣整除。";
    else if(n%3==0 && n%5==0 && n%7!=0)
        cout << n << "␣能被两个数␣(3,␣5)␣整除。";
    else if(n%3==0 && n%5!=0 && n%7==0)
        cout << n << "␣能被两个数␣(3,␣7)␣整除。";
    else if(n%3!=0 && n%5==0 && n%7==0)
        cout << n << "␣能被两个数␣(5,␣7)␣整除。";
    else
        cout << n << "␣不能被␣3,␣5,␣7␣整除。";
    cout << endl;
}

int main()
{
    int n;

    while(1)  // 设置循环，以便重复执行下面的程序段
    {
        cout << "\n␣请输入一个整数（负数退出）：␣";
        cin >> n;
        if(n<0) break;
        f(n);
    }
    return 0;
}
```

【方法二】先完成所有的判断，然后输出，使计算与输出两个环节分离。为此，我们设计一个标志量（一个整型变量 flag），记录 n 被 3, 5, 7 整除的情况。我们**约定**：这个标志变量的低 3 位（从最低位起）分别表示整数 n 被 3, 5, 7 整除的情况（用 0 表示不能整除，用 1 表示能整除），其余的位全置 0。共有 $2^3 = 8$ 种情形。这样，若标志值为 5（二进制为 101 ），则表示整数 n 能被 3, 7 整除，但不能被 5 整除。程序如下。

源代码 4.10 整除问题（推荐方法）

```
1  // 整除问题2.cpp
2  #include <iostream>
3  using namespace std;
4  int mod_3_5_7(int n)
5  {
6      int flag = 0;
7      if(n%3 == 0) flag += 1;       // 二进制 001 或 flag |= 1
8      if(n%5 == 0) flag += 2;       // 二进制 010 或 flag |= 2
9      if(n%7 == 0) flag += 4;       // 二进制 100 或 flag |= 4
10     return flag;
11 }
12 void show_result(int n)
13 {
14     switch(mod_3_5_7(n))
15     {
16     case  0: cout << n << "␣不能被␣3,␣5,␣7␣整除。";    break;
17     case  1: cout << n << "␣仅能被一个数␣(3)␣整除。";  break;
18     case  2: cout << n << "␣仅能被一个数␣(5)␣整除。";  break;
19     case  3: cout << n << "␣能被两个数␣(3,␣5)␣整除。"; break;
20     case  4: cout << n << "␣仅能被一个数␣(7)␣整除。";  break;
21     case  5: cout << n << "␣能被两个数␣(3,␣7)␣整除。"; break;
22     case  6: cout << n << "␣能被两个数␣(5,␣7)␣整除。"; break;
23     default: cout << n << "␣能同时被␣3,␣5,␣7␣整除。";
24     }
25     cout << endl;
26 }
27 int main()
28 {
29     int n;
30     while(1)
31     {
32         cout << "\n␣请输入一个整数（负数退出）：␣";
33         cin >> n;
34         if(n<0) break;
35
36         show_result(n);
37     }
38     return 0;
39 }
```

第二个程序中之所以将**计算**与**输出**分离，是因为有可能采用与 cout 不同的其他输出方法，此时，只要修改输出部分的语句，计算部分的语句不变；或者计算结果有其他用途而根本不做输出操作。

程序第 14 行直接用函数的返回值（整型临时变量，用函数调用表达式本身表示）作为 switch 语句的入口表达式，其值只能是 0～7 之间的整数。最后的 default 情形后使用或者缺省 break; 都是等效的。

4.3.2 三角形的周长及面积

例 4.7 给定三条线段的长度 a, b, c，判断它们是否能够组成一个三角形，若能则计算并“返回”该三角形的周长和面积，否则令其周长和面积全为零。

我们知道，三角形的周长（perimeter）和面积（area）分别为

$$\text{perimeter} = a + b + c$$
$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

其中 $s = (a + b + c)/2$。

【分析】设计**一个**函数完成该项计算。该项计算涉及的数据有三条线段的长度，周长，面积，辅助量 s 共六个数据。其中线段的长度单向传递给函数即可；周长和面积需要由函数传回给函数调用者；辅助量在所论问题中不是独立的量，不必在函数之间来回传递，s 应该作为函数的局部变量。

源代码 4.11 三角形的周长及面积

```
#include <cmath>
int triangle(double a, double b, double c,
             double &perimeter, double &area)
{
    double s = (a + b + c)/2;
    if(s>a && s>b && s>c)
    {
        perimeter = a + b + c;
        area = sqrt(s*(s-a)*(s-b)*(s-c));
        return 1;
    }
    perimeter = area = 0;
    return 0;
}
```

我们为本函数设计了一个返回类型：若 a，b，c 能构成一个三角形，则函数返回 1 （相当于逻辑值 true），否则返回 0（相当于逻辑值 false）。这样的处理几乎不增加函数的设计工作量和难度，却给使用该函数的程序员带来便利：程序员不必在函数调用后通过判断近似表示的浮点型数据 area 是否精确地等于 0 来获知 a，b，c 能否构成一个三角形。

第 3.2.1 小节中所介绍的求解一元二次方程的函数 int Solver(double a, double b, double c, double &x1, double &x2); 与上述函数是类似的。

4.4 单变量版“评委评分”程序设计

从本节开始，将在后续的若干章节中逐步由浅入深地讨论所谓的“评委评分”程序设计。随着这个具体例子的不断扩充逐步介绍 C++ 的语法及程序设计技术，目的是使读者了解相关的 C++ 语法为什么需要（必要性）以及如何使用的问题。

4.4.1 问题描述及算法分析

例 4.8 【“评委评分”问题的提出】有 n 名选手参加的某大奖赛，组委会聘请了 m 个评委为每一位参赛选手评分。每位选手在其所得的 m 个分数中按“去掉一个最高分，去掉一个最低分，剩下的 $m-2$ 个分数的算术平均分”确定最后得分。现要求编写一个程序，帮助大奖赛组委会计算并输出各选手的最后得分。

【算法分析】我们认为，计算机解题的过程与算盘解题过程是一脉相承的。设想这样的情景：大奖赛组委会决定请读者您充当竞赛现场的秘书，组委会仅提供一把算盘。那么如何处理这项工作呢？

（1） 这是一项重复劳动，对每一位选手的数据进行相同的处理（重复次数等于参赛选手人数 n ）。

（2） 对于每一位参赛选手，需要：

- 将算盘上的档分成五段，分别用于存放分数（x）、最高分（max）、最低分（min）、总分（sum）和平均分（aver）。并且对每一位选手计算前都需要先将算盘盘面清零。
- 依次读取各位评委为当前选手所亮出的分数；边读边记录（x），边比较以得到目前的局部最高分（max）、局部最低分（min）及总分的部分和（sum）。这一过程也是重复劳动（重复次数等于评委人数 m ）。当读完全部评委所亮的分数后，这些局部最高分、局部最低分、部分和，便演变成了该选手的最高分、最低分、总分。
- 计算 aver = (sum-max-min)/(m-2) 并宣布当前选手的最后得分。

其中求最高分、最低分的过程就如同“擂台赛”，下一位评委所亮分数与“擂主”比较，更大（或更小）者成为新的“擂主”，因而俗称这种处理方法为**“打擂台算法”**。

4.4.2 程序实现

1. 变量设计

实际问题中的数据需要以变量的形式存放在计算机内存中。另外还需要一些起辅助作用的变量。解决本问题所需要的主要变量如表 4-1 所示。

表 4-1 单变量版“评委评分”程序的变量设计

变量名	数据类型	存储类型	作　　用
referees	int	auto	存放评委人数
players	int	auto	存放参赛选手人数
x	double	auto	存放评委给参赛选手的分数
max	double	auto	存放选手的最高分
min	double	auto	存放选手的最低分
sum	double	auto	存放选手的总分
aver	double	auto	存放选手的最后得分

程序的大致结构是二重循环，外层循环对应于参赛的各选手；内层循环对应于各评委给当前选手评分，以及计算该选手的最高分、最低分、总分，为计算最后得分做准备。

程序运行时，若输入的评委人数为 10，参赛选手人数为 20，则共有 $10 \times 20 = 200$ 个原始分数。如此多的数据，若皆从键盘输入则是十分不便的。为此，程序中利用伪随机数发生器产生各个原始数据，并且将所产生的随机数控制在 8.0 ～ 10.0 之间且仅有一位小数。

2. 源程序代码

源代码 4.12 单变量版“评委评分”程序

```
// contest0.cpp      评委评分（单变量版）
#include <iostream>
#include <ctime>
#include <conio.h>                          // 参见第 2.5 节
using namespace std;
int main()
{
    int referees, players;
    double x, min, max, sum, aver;

    cout << "请输入评委人数:␣";
    cin >> referees;
    if(referees<=2)
    {
        cout << "评委人数太少。" << endl;// 无法继续运行
        return -1;          // 返回 -1 给操作系统表示出现了某种状况
    }
    cout << "请输入参赛选手人数:␣";
    cin >> players;

    time_t t;
    srand(time(&t));
    for(int i=0; i<players; i++)
    {
        cout << "\n\n====␣No.␣" << i+1 << endl;
        min = 10;                           // 最低分一定不超过 10
        max = sum = 0;                      // 最高分一定高于 0
        for(int j=0; j<referees; j++)
        {
            x = (80 + rand()%21)/10.0; // 产生8.0～10.0之间随机数
            cout << x << '\t';              // 取代键盘输入语句 cin>>x;
            if(x > max) max = x;            // 打擂台算法求最大值
            if(x < min) min = x;            // 打擂台算法求最小值
            sum += x;                       // 计算总分
        }
        aver = (sum-max-min)/(referees-2);

        cout << "\n␣去掉一个最高分␣" << max
```

```
39                 << "␣分,␣去掉一个最低分␣" << min
40                 << "␣分，最后得分␣" << aver << "␣分。" << endl;
41             getch();
42         }
43         return 0;                                  // 返回 0 给操作系统表示正常
44     }
```

本程序中没有定义全局变量，因此程序开始执行时直接执行主函数 main。

（1） 首先定义两个整型变量和 5 个双精度浮点型变量（意味着分配内存空间）。

（2） 第 11 行为输出提示信息，以指引操作员在接下来的输入语句中正确地输入。倘若没有适当的提示信息，操作员将可能无所适从。

（3） 执行第 12 行语句时，系统将暂停、等待操作员输入数据并以回车结束输入。倘若输入的数值太小，则将会输出一个信息指出：“评委人数太少。”，进一步遇到 return -1; 语句而终止程序，将 -1 返回给操作系统。操作系统或者用户可以检测该程序的返回码（此处为 -1）根据事先的约定以了解程序的执行情况，程序结束的原因。

（4） 若输入的评委人数大于 2，则继续执行第 18 行的语句，输出提示信息；及第 19 行的语句：暂停、等待操作员输入参赛选手人数并回车。

（5） 程序第 23～42 行为外层循环，循环控制变量 i 的**初值**为 0；**步长**为 1（即每循环一次，循环控制变量 i 的值增加 1）；终值为 players-1。外层循环将执行 players 次。循环结束时 i 的值为 players，即继续循环的条件不再成立之时。

（6） 立足循环体内（第 25～41 行）看变量 i，则 i 是暂时不变的，这里特指第 i+1 位参赛选手。第 26～27 行便相当于“算盘”盘面上的清零操作：为了计算最低分，我们首先在“擂台”上放置一个最大值（本例中 10.0 为最大值）；为了计算最高分，“擂台”上放置了一个低于 8.0 的分数 0；计算总分的**累加器** sum 清零，这都是合理的或必需的。

（7） 第 28～35 行为内层循环，其功能是处理第 i+1 位选手的分数。其中，第 30 行产生一个随机数并变换成 8.0～10.0 之间的数，且**仅包含一位小数**。内层循环结束后，便计算该选手的“最后得分”（第 36 行），然后输出该选手的有关信息。

大赛结束后，“算盘”上留下来的数字应该有：x 是最后一位评委给最后一名选手的分数， max，min，sum，aver 是最后一名选手的相关数据。上述程序结束前，这些变量的值与“算盘”上的数值一样。此时变量 j 的值等于 referees，变量 i 的值等于 players。

若 referees 取 10，players 取 20，则有 200 个原始分数都存放在同一内存单元 x（反复覆盖前一个数据值）；也分别各仅用一个变量存放过各位选手的最高分、最低分、总分和最后得分。这就是我们将该程序称为“单变量版”程序的原因。遗憾的是，前 199 个原始分数，以及前 19 位选手的所有信息都不存在了（注：显示器属于外部设备，虽然此时在显示器上仍然显示着有关数据，但是计算机内存中已经只有最后少量的几个数据了）。

我们知道，计算机是一个拥有巨量“档”数的“算盘”，有大容量的内存供我们支配使用。如何改进上述程序，使所有原始数据得以保留以便再利用，且看下章分解。

4.5 * 趣味程序——击打字母游戏

例 4.9 击打字母游戏。在显示器第二行中部若干列的随机位置、随机地出现一个大写或小写英文字母。操作员在规定的延时时间（ delay =2000 ms）内在键盘上击打对应的键，击打正确则可得分。可修改本程序使得分及格者过关（每关 20 个字母），每过一关则适当缩短延时时间。

源代码 4.13 击打字母游戏

```
// typing.cpp            击打字母游戏
#include <iostream>
#include <iomanip>              // 为了使用 setw
#include <ctime>
#include <conio.h>
using namespace std;
int hitted(int n, int delay)  // 共被要求击 n 个字符，延时 delay 毫秒
{                             // 返回正确击打字母的个数
    int i, m=0, w;
    char ch, key;
    clock_t t;
    cout << endl;
    for(i=0; i<n; i++)
    {
        ch = 'A' + rand() % 26;           // 随机产生一个大写字母
        if(rand() % 2) ch += 'a' - 'A';   // 可能被转换成小写字母
        w = rand() % 40 + 20;             // 随机产生输出位置

        cout << setw(w) << ch << '\r';    // 输出字符

        t = clock() + delay;              // 确定延时终止时间
        while(clock() <= t)               // 延时循环
        {
            if(kbhit())                   // 判断是否有击键
            {
                key = getch();            // 接收键盘输出
                if(key == ch)             // 判断是否正确
                    m++;                  // 累计正确字符数
                else
                    cout << "\a\r";       // 不正确时响铃
                break;                    // 不等待延时期满
            }
        }
        cout << setw(w) << '␣' << '\r';   // 清除原来的字符
    }
    return m;
}
```

```
38
39 int main()
40 {
41     int n = 10, delay=2000, m;           // 可修改 n, delay 的值
42     time_t t;
43     srand(time(&t));
44
45     m = hitted(n, delay);
46
47     cout << "\n" << m << "/" << n << "␣=␣"
48          << 100.0*m/n << "%" << endl;
49
50     cout << "Press␣<ESC>␣..." << endl;
51     while(getch()!=27);                  // ESC 键的编码为 27
52     return 0;
53 }
```

4.6 小 结

计算机内存是数据的载体，变量是计算机内存的抽象。计算机高级语言程序通过变量使用计算机内存，同时通过变量刻画所计算的问题。程序设计中，变量设计是关键，变量所代表的实际量的逐步变化代表着问题求解过程的递进。我们必须关注变量的值的变化规律——初始化值、当前值、最终值。每个变量都有自己独立的运行轨迹，把握了变量的变化规律就能很好地把握程序。我们称变量是程序中的“小精灵”。程序员是这些“小精灵”们的主人：创造它们、给它们住所、为它们命名、规定它们的角色，等等。控制好这些“小精灵”们是程序员的职责，也能体现程序员控制、驾驭它们的能力。

对于一个命题作文，不同的人写出不同的作品。程序设计也一样，对同一问题可以有多种不同的算法、不同的方案来解决。读者应该善于总结基本的程序设计方法和技巧，做到举一反三。

练 习 4

1. 指出下列各程序或函数中的错误

（1）希望判定整数是否为素数。

```
1  int isprime(unsigned int n)
2  {
3      unsigned int k;
4      for(k=2; k<n; k++)
5      {
6          if(n%k==0)
7              return 0;
8          else
9              return 1;
10     }
11 }
```

（2）希望根据 $\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots$ 计算圆周率 π 。

```
double pi()
{
    int n=1;
    double s=0, p=1, epsilon=1e-8;
    do
    {
        s += p;
        n += 2;
        p *= -1/n;
    }while(p>=epsilon || p<=-epsilon);
    return 4*s;
}
```

（3）希望根据 $e^x\approx 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots+\frac{x^{20}}{20!}$ 以级数前 20 项之和为 e^x 的近似值。

```
double Exp20(double x)
{
    double s=0, p;
    int i, n, q;
    for(n=1; n<=20; n++)
    {
        p = q = 1;
        for(i=1; i<=n; i++)
        {
            p *= x;  q *= i;
        }
        s += p/q;
    }
    return s;
}
```

（4）希望根据二分法求解方程 $f(x)=e^{-x}-(100x)^{-3}=0$ 的在区间 $(1,50)$ 之间的根（事实上 $f(1)>0, f(50)<0$，精度较高的解为 $23.2551, f(23.2551)=9.5163\times 10^{-20}$）。

```
#include <iostream>
#include <cmath>
using namespace std;

double f(double x){ return exp(-x)-pow(100*x, -3); }

double g1(double a, double b)
{
    double c;

    if(a>b)
    {
        c=a; a=b; b=c;          // 使 a 为区间左端点
```

```
    }

    do
    {
        c = (a+b)/2;
        if(f(c)==0) break;
        if(f(a)*f(c)<0) b = c;
        else            a = c;
    }while(fabs(f(c))>=1e-8);
    return c;
}

double g2(double a, double b)
{
    double c;
    if(a>b)
    {
        c=a; a=b; b=c;          // 使a为区间左端点
    }
    while(true)
    {
        c = (a+b)/2;
        if(f(c)==0) break;
        if(f(a)*f(c)<0) b = c;
        else            a = c;
    }
    return c;
}

int main()
{
    cout << g1(1, 50) << endl;  // 输出25.5
    cout << g2(1, 50) << endl;  // 执行函数陷入无穷循环无法返回
    return 0;
}
```

（5）希望求两个正整数的最大公因数及最小公倍数。

```
void gcd_lcd(unsigned int m, unsigned int n
             unsigned int &gcd, unsigned int &lcm)
{
    unsigned int temp;
    while(n)
    {
        temp = n;  n = m%n;  m = temp;
    }
    gcd = m;
    lcm = m*n/gcd;
}
```

（6）给定 4 个数据类型相同的变量（如 double a1, a2, a3, a4;），希望将它们的值进行交换使其满足 $a_1 \leqslant a_2 \leqslant a_3 \leqslant a_4$ （即对它们进行从小到大排序）。

```
void Sort(double a1, double a2, double a3, double a4)
{
    double temp;
    if(a1 > a2){ temp = a1; a1 = a2; a2 = temp; }
    if(a2 > a3){ temp = a2; a2 = a3; a3 = temp; }
    if(a3 > a4){ temp = a3; a3 = a4; a4 = temp; }

    if(a1 > a2){ temp = a1; a1 = a2; a2 = temp; }
    if(a2 > a3){ temp = a2; a2 = a3; a3 = temp; }

    if(a1 > a2){ temp = a1; a1 = a2; a2 = temp; }
}
```

2. 基本题

（1） 阅读程序，分析运行结果。

```
// Fibonacci.cpp
#include <iostream>
using namespace std;

int main()
{
    int a=1, b=1, i;

    cout << a << ",␣" << b;
    for(i=3; i<16; i+=2)
    {
        a = a + b;
        b = a + b;
        cout << ",␣" << a << ",␣" << b;
    }
    cout << endl;
    return 0;
}
```

（2） 编程计算 $1! + 2! + \cdots + 15!$，要求输出每个部分和的值。

（3） 编程求 1000 之内的所有“完数”。“完数”是指一个数恰好等于它的所有因数（不包括该数本身）之和。例如，6 是完数，因为 6 的因数有 1，2，3，6 且 1+2+3=6。

（4） 编写一个函数原型为 int f(int n); 的函数，对于给定的正整数 n 计算并返回不超过 n 的被 3 除余 2， 并且被 5 除余 3，并且被 7 除余 5 的最大整数，若不存在则返回 0。

（5） 设 $f(x)=\cos(x)-x$，容易验证 $f(0)f(1)<0$，根据连续函数的性质可知在区间 $(0,1)$ 中，方程 $f(x)=0$ 至少有一个根。编程用二分法求其一个根，要求精度为 10^{-8}。

二分法的计算过程：计算区间中点 c 的函数值，若 f(c) 为 0，则 c 即为所求；否则判断 f(a) 与 f(c) 是否异号，若它们异号则去掉右半区间（即令区间右端点 b=c），否则去掉左半区间（即令区间左端点 a=c），直到区间长度小于预先给定的精度控制值（如 $\varepsilon=10^{-8}$）。

（6） 计算 $\sqrt[3]{x}$ 的迭代公式为

$$\begin{cases} y_0=1 \\ y_{n+1}=y_n+\left(\dfrac{x}{y_n^2}-y_n\right)/3 \qquad n=1,2,\cdots \end{cases}$$

试编写一个函数 `double cuberoot(double x);` 计算浮点型数值的立方根，并编写一个主函数测试它（包括计算正数、零、负数的立方根）。

（7） 编程求解问题：有一种细菌，从其产生的第 4 分钟后，每分钟都产生一个同种细菌。若某初始时刻仅有一个这种细菌，那么此后第 n 分钟时共有多少个这种细菌？

提示：将产生后 1, 2, 3 分钟及大于等于 4 分钟的细菌数分别记为 a，b，c，d，然后分析其变化规律。

3. 扩展题（程序设计竞赛题型）

判断算式正确性。

问题描述 给定一个算式，该算式中只含一个四则运算符号，操作数及结果均为整数。要求判断该算式的正确性（规定：除法必须除尽才可能正确）。

输入说明 输入数据有多行，每行为一个算式。

输出说明 对于每一种情形，要求先输出“Case 序号：”，然后输出 correct（表示算式正确）或者 incorrect（表示算式错误）。

输入样例（有意不将算式编排整齐）

```
1+2 = -3
1 - 2 = -1
5/ 2 = 2
4 /2=2
```

输出样例

```
Case 1: incorrect
Case 2: correct
Case 3: incorrect
Case 4: correct
```