C++ 程序设计习题及解答

C++ 程序设计系列教材

钱 能 主编

清 华 大 学 出 版 社
http://www.tup.tsinghua.edu.cn

C++ 程序设计习题及解答

钱 能 编

清华大学出版社

(京) 新登字 158 号

内 容 简 介

本书是《C++ 程序设计教程》的配套教材，内容包括 67 例C++ 程序设计习题解答，其中每一例习题解答都是精选的讲课例子，系统地向读者解答了有关程序设计中具有一定代表性的问题。既可帮助计算机爱好者提高编程能力，又为开发软件的程序员提供了编程方面的素材。

书　　名：C++ 程序设计习题及解答
作　　者：钱　能
出 版 者：清华大学出版社（北京清华大学学研大厦，邮政编码：100084）
　　http://www.tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
发 行 者：新华书店总店北京科技发行所
开　　本：787 × 1092　1/16　印张：10　字数：203 千字
版　　次：1999 年 12 月第 1 版　2001 年 3 月第 7 次印刷
书　　号：ISBN 7-302-02140-6/TP·2198
印　　数：41001 ~ 51000
定　　价：14.00 元

前　言

学习程序设计方法，并且要实质性地提高编程能力，除了看书理解之外，有两个关键的因素：一个是做编程书面练习，一个是上机做实验。在初级程序设计的能力培养中，唯有多练，才能真正找到编程的感觉，才能培养出宝贵的编程经验。这些感觉和经验，一定程度上决定了日后成为什么样的计算机人才。所有计算机高级人才，都曾走过风风雨雨的编程历程。

本书既为那些想提高编程能力的初学者准备，又为设计、开发软件的程序员提供了编程方面的素材。书中共有习题67例，每个习题解答都是精选的讲课例子，对程序设计中存在的问题具有一定的代表性。有些解答还提供了不止一种方法。书中除了个别问答题外，全部解答都通过BC++5.0的调试和运行，并配有运行结果，以便读者在不同环境下运行比较。

程序的解答可以各种各样，越吃透程序结构、吃透语言的内在含义和联系，就越能准确表达解决问题的方法，编制的程序也就越简练，对日后深入学习程序设计理论的领悟性也越高。读者可以先自己尝试解答，然后再参考书中解答。书中的解答与系列丛书《C++程序设计教程》中的习题具有相似性，可以起到提示和帮助读者解答教程中习题的作用。当然书中的解答并不一定是最优的，它们只是具有代表性而已。欢迎读者提出自己的见解，编出更高质量的程序。

虽然这只是迈出了一小步，但作者愿与程序设计爱好者们一起努力，在明年的再版中，更上一层楼。

作者的学生刘雪芬、沈雪飞和徐晓萍做了大量习题中程序的录入和调试工作，在此表示深深的感谢。

作者的电子邮件地址是：

qianneng@mail.hz.zj.cn

通信地址是：

杭州市德胜新村12幢1单元101室　钱　能　收　　（邮编310014）

目　　录

过程化基础

1. 三个简单的求值计算

分别输入圆柱体的半径和高，求其体积；输入球半径，求其表面积；输入长方体的长、宽、高，求其体积。

[解答]

解法一：由于其中的三个问题是相互独立的，所以，解决方法也是依次来解决这三个问题。先输入第一个问题的半径和高，计算并输出圆柱体的体积；然后输入球半径，计算球面积并输出；然后输入长、宽、高，计算长方体的体积并输出。

```
//- - - - - - - - - - - - - - - - - - - - - - - -
 #include <iostream.h>
 #include <math.h>
//- - - - - - - - - - - - - - - - - - - - - - - -
void main()
{
  //求圆柱体的体积
  double radius, height;
  cout <<"请输入圆柱体的半径和高:\n";
  cin >>radius >>height;

  double volume=radius * radius * height * M_PI;

  cout <<"该圆柱体的体积为:" <<volume <<endl;

  //求球面积
  cout <<"\n请输入球半径:\n";
  cin >>radius;

  double areaOfSphere=4 * radius * radius * M_PI;

  cout <<"该球面的面积为:" <<areaOfSphere <<endl;

  //求长方体的体积
  double length,width;
  cout <<"\n请输入长方体的长、宽、高:\n";
  cin >>length >>width >>height;

  volume=length * width * height;

  cout <<"该长方体的体积为:" <<volume <<endl;
```

```
}
//- - - - - - - - - - - - - - - - - - - - - - -
```

运行结果为：

```
请输入圆柱体的半径和高：
32  44
该圆柱体的体积为:141548

请输入球半径：
30
该球面的面积为:11309.7

请输入长方体的长、宽、高：
32  12  15
该长方体的体积为:5760
```

π是常量，它在C++中有专门的表示，可以利用之，其名称为M_PI，在math.h的头文件中，因此程序要包含之。

程序中的变量名radius在求圆柱体和球面积时都用到了它，同样，height也用了两次，变量名可以被反复赋值。

运行结果中，double类型的数值默认只有6位精度，所以圆柱体的体积没有小数部分了。

解法二：对于一个复杂的计算问题，也可以采用分而治之的方法，即用函数调用分别完成预定的“小任务”，返回其计算结果。当然这些“小任务”要预先声明，若要能运行，还必须有其定义：

```
//- - - - - - - - - - - - - - - - - - - - - - -
 # include < iostream.h >
 # include < math.h >
//- - - - - - - - - - - - - - - - - - - - - - -
double Cylinder(double r, double h);
double Sphere(double r);
double Rectangle(double l, double w, double h);
//- - - - - - - - - - - - - - - - - - - - - - -
void main()
{
  double radius, height;
  cout <<"请输入圆柱体的半径和高: \ n";
  cin >> radius >> height;

  double volume = Cylinder(radius, height);

  cout <<"该圆柱体的体积为:" << volume << endl;

  cout <<"请输入球半径: \ n";
  cin >> radius;

  double areaOfSphere = Sphere(radius);
```

```
    cout <<"该球面的面积为:" << areaOfSphere << endl;

    double length,width;
    cout <<"请输入长方体的长、宽、高:\n";
    cin >> length >> width >> height;

    volume = Rectangle(length,width,height);

    cout <<"该长方体的体积为:" << volume << endl;
}
//-----------------------
double Cylinder(double r, double h)
{
   return r * r * M_PI * h;
}
//-----------------------
double Sphere(double r)
{
  return 4 * r * r * M_PI;
}
//-----------------------
double Rectangle(double l, double w, double h)
{
  return l * w * h;
}
//-----------------------
```

该程序与上一个程序的功能一样。但是分别将三个具体的计算工作挪到另外的函数中去做了。函数最基本的模式就是接受一些参数作为输入数据,经过自身的计算,返回其计算结果。从程序的组织来看,这样得到的程序并不比原来的程序更简单和更易理解,只有在计算过程比较复杂的情况下,将该计算部分分离出去才是合理的。

我们也可以将一个程序看作是由若干个过程组成的,一个过程是输入、处理和输出的整体。这样就得到下面的程序:

```
//-----------------------
#include <iostream.h>
#include <math.h>
//-----------------------
void Cylinder();
void Sphere();
void Rectangle();
//-----------------------
void main()
{
  Cylinder();
  Sphere();
  Rectangle();
}
//-----------------------
```

```
void Cylinder()
{
  double radius, height;
  cout <<"请输入圆柱体的半径和高:\n";
  cin >>radius >>height;

  double volume = radius * radius * M_PI * height;

  cout <<"该圆柱体的体积为:" <<volume <<endl;}
//- - - - - - - - - - - - - - - - - - - - - -
void Sphere()
{
  double radius;
  cout <<"\n请输入球半径:\n";
  cin >>radius;

  double areaOfSphere = 4 * radius * radius * M_PI;

  cout <<"该球面的面积为:" <<areaOfSphere <<endl;
}
//- - - - - - - - - - - - - - - - - - - - - - - -
void Rectangle()
{
  double length,width,height;
  cout <<"\n请输入长方体的长、宽、高:\n";
  cin >>length >>width >>height;

  double volume = length * width * height;

  cout <<"该长方体的体积为:" <<volume <<endl;
}
//- - - - - - - - - - - - - - - - - - - - - - - -
```

该程序的功能也和上面一样,C++ 把过程也看做为函数。在这里可以看到,主函数简单了,读这个程序,直截了当地知道它要处理三件事。根据函数名 Cylinder(),Sphere(),Rectangle(),可以大概猜到该三个过程干什么事。然后在接下来的函数定义中,彻底知道其功能。这种程序设计思想是我们解决不太大问题的最基本出发点。可以传递给函数以参数,返回函数值;可以不传递参数而让其返回值;可以传递以参数而不需返回值;也可以既不传递参数,又不返回值。在具体程序设计中,由程序员视问题的特点而定。从上面的例子中看到,不传递参数且不返回值的功能划分比较理想,它使得程序比较简洁明快。

请自己动手在主函数中输入变量的值,分别传输参数给三个函数,让函数完成计算面积或体积并打印结果的工作(不返回值)。

又请在主函数中调用三个无参函数(不传递参数给函数),分别返回计算结果,在主函数中,打印之。

2. 格式化显示对数值

编写一个程序,打印 90 ~ 100 之间数的自然对数与以 10 为底的对数。对表进行格式

化，使得数字可以在域宽为 10 的范围内，用 5 个十进制位置的精度进行右对齐。

[解答]

输出长度格式化可以用 cout.width(n)来指定长度为 n 的数据表示，它作为一种函数调用而单独成句。

```
//- - - - - - - - - - - - - - - - - - - - -
#include <iostream.h>
#include <math.h>
//- - - - - - - - - - - - - - - - - - - - -
void main()
{
  cout.precision(5);    //5 位有效位，非小数点后位数
  cout <<"        x       logx        lnx \n \n";
  for(int x=90; x<100; x++){
    cout.width(10);
    cout << x;
    cout.width(10);
    cout << log10(x);
    cout.width(10);
    cout << log(x) << endl;
  }
}
//- - - - - - - - - - - - - - - - - - - - -
```

运行结果为：

```
 x         logx         lnx
90       1.9542       4.4998
91       1.959        4.5109
92       1.9638       4.5218
93       1.9685       4.5326
94       1.9731       4.5433
95       1.9777       4.5539
96       1.9823       4.5643
97       1.9868       4.5747
98       1.9912       4.585
99       1.9956       4.5951
```

如果包含了头文件 iomanip.h，则 cout << setw(10)能起到同样的效果。

3. 求 n! 末尾 0 的个数

在长整数表示范围内的 n，求 n! 中末尾 0 的个数。

[解答]

一般解法是先求 x = n! 的值，然后不断除 10 取余。比如：12! 等于 39916800。在第三次除 10 取余时，发现余数不为 0，因而，得出结论，12! 中，末尾 0 的个数为 2。

注意所求值的数据类型应为 long int,因为要表示尽可能大的整数值。初步求得算法为:

1. 给出 n! 的 n
2. X←[n!]　　//求 n! 由一个函数完成
3. I←0　(I 为尾数 0 的计数)
4. FOR(;X%10==0;I++)
5. 　X←X/10
6. 输出 I

对应的程序源代码为:

```
//- - - - - - - - - - - - - - - - - - - - - - - -
#include <iostream.h>
//- - - - - - - - - - - - - - - - - - - - - - - -
long fact(long n); //求 n!
//- - - - - - - - - - - - - - - - - - - - - - - -
void main()
{
  long x,n;
  cout <<"请输入一个正整数:";
  cin >> n;     //n 应大于 0

  x = fact(n);

  int i = 0;       //i 须在循环外定义,因为循环之后要用到它
  for( ; x % 10 == 0; i ++ )
    x /= 10;

  cout << n <<"!中末尾 0 的个数为" << i << endl;
}
//- - - - - - - - - - - - - - - - - - - - - - - -
long fact(long n)
{
  long s = 1;
  for(int i = 1; i <= n; i ++ )
    s * = i;

  return s;
}
//- - - - - - - - - - - - - - - - - - - - - - - -
```

但是,long int 型整数最大只能表示 12!,13! 以上将得不到正确结果。所以当 n! 之 n 超过 12 时,靠上述的办法无法求得末尾 0 的个数。题意也并没有非要先求 n! 再求末尾 0 的个数不可。

分析求 100! 的末尾 0 的个数:

末尾有几个 0,即有几个 10 的因子。而 $10 = 2 \times 5$。从 1 ~ 100 所有的整数乘积中,2 的因子比 5 的因子多,因此只要算 100! 中含有因子 5 的个数即为含有因子 10 的个数。在 1 ~ 100中,含有因子 5 的数出现了 20 次,其中有四个数(25,50,75,100)中各含有两个 5 的因

子,因此总共含有因子5的个数为20+4=24,故100!中,末尾含有24个0。

从中得到启发,求n!中末尾0的个数,只要求得n!中因子5的个数,其中,可能有些数含有多个5的因子。

于是有以下算法:

(1) 给出n!的n

(2) K←0 (K为尾数0的计数)

(3) FOR(I从5→n,步长5)

(4)　X←[I中含有因子5的个数]

(5)　K←K+X

(6) 输出K

[I中含有因子5的个数]之算法为:

(1) M←0

(2) For(;I%5==0; M++)

(3)　I←I/10

(4) 返回M

对应的程序源代码为:

```
//- - - - - - - - - - - - - - - - - - - - - - -
//////////////////////////////////////////
//              fact0.cpp                //
//////////////////////////////////////////
//- - - - - - - - - - - - - - - - - - - - - - -
#include <iostream.h>
//- - - - - - - - - - - - - - - - - - - - - - -
long numOf5(long n); //求因子5的个数
//- - - - - - - - - - - - - - - - - - - - - - -
void main()
{
  long n;
  cout <<"请输入一个正整数:";
  cin >>n; //n应大于0

  long k=0;
  for(int i=5; i<=n; i+=5) //对从i~n的每个数,求因子5的个数之和
    k += numOf5(i);

  cout <<n <<"!中末尾0的个数为" <<k <<endl;
}
//- - - - - - - - - - - - - - - - - - - - - - -
long numOf5(long n)
{
  long i=0;
  for( ; n%5==0; i++) //如果一开始i不能整除5,则返回的i为0
    n/=5;

  return i;
```

```
}
//- - - - - - - - - - - - - - - - - - - - - - - - -
```

运行结果为：

```
请输入一个正整数:1000
1000!中末尾 0 的个数为 249
```

设计算法时，并不一定要所有步骤都详细列出，遇到一下子求解有困难的步骤，可以先用文字作简单(抽象)描述，把它看做是一个函数，如上例中的[I 中含有因子 5 的个数]。等到整体过程描述完后，再来具体描述其中的函数。

上例的算法并不一定是最好的，还能找到更好的解。更好的解，一般首先用程序语句的简洁性和可读性来衡量，其次是用可移植性、可维护性，以及高效性衡量。这些程序质量好坏的因素，是综合考虑的。

程序设计中所用的方法，一般都是从具体分析中获得，如上例中先求 12!，看到尾数有两个 0。具体分析总是从一个比较简单的实例开始。先手工求得解，掌握求解的步骤，归纳出一般的程序设计方法。

算法与程序实现之间还不能完全一一对应，因为程序实现受程序语言表达能力的限制。上例中，由于数据类型 long int 表示范围的限制，使得实现中不得不另求出路。其最后解决方案也是从分析具体实例中得出，如上例中分析 100! 的末尾 0 的个数。最后上机调试是必须的，用几个有代表性的数据验证，才能基本肯定一个问题的解答。

4. 求 n!

对整数范围内的 n，求 n!。

[解答]

由于 C++ 程序设计语言中的基本数据类型所表达的数值范围有限，对于一个比较大的 n(比如 n = 100)，求 $n \times (n-1) \times (n-2) \times \cdots \times 1$ 时，表达其乘积的变量值必定要溢出。所以，单纯用一个循环求解 n! 的算法行不通。

可以把 n! 的结果放在数组中，数组中的每个元素都表示其 n! 值的一位。

对于输入的 n，想办法尽量精确地估计出其 n! 所占的位数，就能确定数组元素的个数。

可以将 n! 表示成 10 的次幂，即：$n! = 10^M$，则不小于 M 的最小整数就是 n! 的位数(此处求 M 的技术使程序不但简化，而且用的空间能尽可能地小)。对该式两边取对数，有：M = log10n!，即：

$$M = \log_{10}1 + \log_{10}2 + \log_{10}3 + \cdots + \log_{10}n。$$

循环求和，就能算得 M 值，该 M 是 n! 的精确位数。

数组空间用申请堆内存来解决。由于每个数组元素表示 n! 的一位数值，所以，数组的类型用占一个字节的 char 型已经足够。

当 n 很大时，堆空间申请有可能失败。所以，要考虑堆空间失败时的执行动作。

数组初始化时，令数组的第一个元素(n! 的第一位)为整数 1，其余为 0。

在数组中计算 n! 时，是通过将数组中的值乘 2，乘 3，乘 4，……一直到乘 n 的方式求得的。把数组的第一个元素看做是 n! 的最低位，最后一个元素看做是最高位。

若当前数组中的内容为 7! =5740(0 是数组的第一个元素，5 是数组第四个元素)，紧接着将乘以 8。这时，8! 的位数(5)将首先被求得，然后，按从个位到高位的顺序逐个乘以 8。乘出的积与上次的进位值相加，由此可以确定当前位的值(即相加值的个位数)，再将该相加值整除 10 后作为下一次的进位值。具体操作如下：

进位值初值为 0，

8×0(个位数)=0，加上进位值得 0，从而确定个位数为 0，进位值为 0；

8×4(十位数)=32，加上进位值得 32，从而确定十位数为 2，进位值为 3；

8×7(百位数)=56，加上进位值得 59，从而确定百位数为 9，进位值为 5；

8×5(千位数)=40，加上进位值得 45，从而确定千位数为 5，进位值为 4；

8×0(万位数)=0，加上进位值得 4，从而确定万位数为 4，进位值为 0；

循环结束。

数组在做 I 从 2 到 n 的乘法中，末尾 0 的个数不断增多，而 0 乘 I 总是为 0。乘 I 时，可以跳过这些 0，因而算法可以作适当的优化。

设置起始位，初值为 0。每次乘 I 前，都判断一下起始位值是否为 0，若为 0，则起始位增 1。每次乘 I，都从起始位开始。

最后，从数组的最后一个元素(n! 的高位)起，直到第一个元素止，逐个打印数组元素的整数值，即为所求 n!。

根据上述内容，得到下面的算法：

(1) [给出 n]

(2) [计算 n! 的位数]

(3) [申请数组空间及初始化]

(4) [求 n!]

(5) [输出 n!]

(6) 返还数组空间

(7) 结束

[给出 n]的算法为：

(1) 输入 n

(2) 若 n 小于 0 则重新输入 n

(3) 若 n 等于零，则结束

(4) 若 n 大于 1，返回 n

[计算 n! 的位数]的算法为：

(1) SUM←1

(2) FOR(I:1→n，步长 1)

(3)　SUM←SUM + $\log_{10}$I

(4) 返回 SUM 的整数部分

[申请数组空间及初始化]的算法为：

(1) 根据 n! 的位数申请堆空间
(2) 若申请失败
(3)　则显示申请失败,并退出程序的运行
(4) 数组第一元素←1
(5) FOR(I:2→N! 的位数,步长 1)
(6)　第 I 个数组元素←0
(7) 返回数组空间首地址

[求 n!]
(1) 1! 位数←1
(2) 起始位←0
(3) FOR(I:2→n,步长 1)
(4)　相加值←0
(5)　I! 位数←(I - 1)! 位数 + $\log_{10}$I
(6)　若起始位值等于零
(7)　　起始位←起始位 + 1
(8)　FOR(J:起始位→I! 位数,步长 1)
(9)　　相加值←相加值 + I * 第 J 个元素值
(10)　　第 J 个元素值←相加值的个位数
(11)　　相加值←相加值整除 10
(12) 结束

[输出 n!]
(1) 位序数←0
(2) FOR(I:数组最后元素位→数组第一元素,步长 - 1)
(3)　若位序数整除 50
(4)　　则换行输出"第几个 50 位:"
(5)　输出第 I 元素的整型值
(6)　位序数←位序数 + 1
(7) 换行
(8) 结束

对应的程序源代码为:

```
//- - - - - - - - - - - - - - - - - - - -
////////////////////////////////////////
//          factor.cpp                //
////////////////////////////////////////
//- - - - - - - - - - - - - - - - - - - -
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
#include <stdlib.h>
//- - - - - - - - - - - - - - - - - - - -
```