C++语言程序设计

实验指导与习题解答

韩　蒙　主　编
詹晓宇　邢惠民　副主编

电子科技大学出版社

C++语言程序设计
实验指导与习题解答

韩 蒙 主 编
詹晓宇 邢惠民 副主编

电子科技大学出版社

图书在版编目（CIP）数据

C++语言程序设计实验指导与习题解答 / 韩蒙主编. --
成都: 电子科技大学出版社, 2016.7
ISBN 978-7-5647-3787-0

Ⅰ. ①C… Ⅱ. ①韩… Ⅲ. ①C 语言－程序设计－高等
学校－教学参考资料 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2016)第 168747 号

C++语言程序设计实验指导与习题解答

主　编　韩　蒙

副主编　詹晓宇　邢惠民

出　　版：电子科技大学出版社（成都市一环路东一段 159 号电子信息产业大厦　邮编：610051）
策划编辑：辜守义
责任编辑：辜守义
主　　页：www.uestcp.com.cn
电子邮件：uestcp@uestcp.com.cn
发　　行：新华书店经销
印　　刷：四川永先数码印刷有限公司
成品尺寸：185mm×260mm　印张 16.5　字数 390 千字
版　　次：2016 年 7 月第一版
印　　次：2016 年 7 月第一次印刷
书　　号：ISBN 978-7-5647-3787-0
定　　价：58.00 元

前　言

C++语言是从C语言发展起来的一种面向对象程序设计语言，与C语言具有良好的兼容性。由于C++的突出优点，使得C++是目前应用最为广泛的面向对象语言之一。

学习高级语言程序设计课程必须重视实践环节。“C++语言程序设计”是一门实践性很强的课程。该课程的学习有其自身的特点，学习者需要通过大量的上机编程训练，在实践中掌握语言知识，训练编程思维，培养学生程序设计的基本能力，并逐步理解和掌握程序设计的思维和方法。

本书是《C++语言程序设计》的配套用书，以培养学生的程序设计能力为编写目的。在结构设计上，结合教材习题强调实践，各实验项目紧密结合相关知识点并根据课程教学学时安排而设计。全书共分4章，内容包括C++上机指南，程序调试和排错，教材习题参考答案以及11个实验指导实例，详细介绍了C++语言程序设计实验指导教材的相关习题解答与实验方法，既适合广大高校计算机专业作为教材选用，也适合广大职业教育培训班选用。

本书具有如下主要特点：

（1）上机实验内容丰富，覆盖面广。根据学校进度安排相应的实验内容，通过习题参考答案和实例实训指导，在调试和排错的过程中对实验中涉及的知识点加以分析，帮助读者顺利完成实践操作。

（2）实验指导实例中通过分析案例的设计思路并给出其中的完整程序代码，对算法及程序各功能模块的解析，使学生了解完整的程序设计调试及排错技巧，掌握C++语言开发项目的基本思路与方法，提高学生的项目实战经验。

本书由韩蒙、詹晓宇、邢惠民共同编写，并由韩蒙老师统稿及修订后形成终稿。囿于编写及出版编校时间所限，不足及疏漏之处，恳请广大读者不吝指正。

编　者

2016年6月

目 录

第 1 章　教材习题参考解答

1.1 第 2 章 C++语法

1. 下述程序的输出为

a = 3, b = 4, c = 2

解释为什么？

```
#include <iostream>
using namespace std;
int     f(int i)       { return ++i; }
int&    g(int &i)      { return ++i; }
int     h(char i)      { return ++i; }
int     main()
{
        int a = 0, b = 0, c = 0;
        a += f(g(a));
        b += g(g(b));
        c += f(h(c));
        cout << “a = “ << a << “, b = ” << b << “, c = ” << c << endl;
        return 0;
}
```

解：

首先我们来分析表达式 a += f(g(a))。式中最先执行的表达式是函数调用表达式 g(a)（此时，a == 0）。请大家注意到函数 g 的形式参数是引用，所以函数的形参 i 可以看成实参 a 的别名（当然，这次的别名绑定是暂时的，仅发生在函数 g 内部）。g 中的语句++i 实际上是对 a 进行自加，这样一来，a 等于 1，同时函数 g 返回了对 a 的引用。此后，返回的引用（实际上就是变量 a）传递给了函数 f。请大家注意，f 的形参是值参，所以这里的 i 只是实参 a 的一个拷贝（此时，i 等于 1）。此后 f 中的++i 只是使 i == 2，同时 f 返回一个值 2，而 a 本身没有变化。以后的问题就简单了，a += 2（执行该语句之前 a == 1）的结果就是 a == 3。

再来分析表示式 b += g(g(b))。内部的 g 函数调用发生的情况同上（此时 b == 1）。只是当内部的 g 返回后，外层的 g 函数有作用在 b 上，使 b 本身发生了变化（此时 b == 2）。再后的 b += b 就使得 b == 4。

最后来分析表示式 c += f(h(c))。式中最先执行的表达式是函数调用表达式 h(c)。请大家注意到 h 的形参 i 的类型是 char，与实参的类型 c(int 型)不符，所以此时 i 接受值是 c 的一个拷贝（此时，i 等于 0）。h 中的++i 使 i 等于 1，同时 h 返回值 1，而实参 c 本身没有变化（此时 c == 0）。以后的 f 调用大家都知道了，它使形参自增，而实参没有变化，同时返回

值 2。最后的 c += 2 使得 c 等于 2。

2. 下列程序的输出是什么？解释输出结果。

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

int& f()
{
    static int  i = 2;
    return ++i;
}

int g()
{
    int j = 2;
    return ++j;
}

int  main()
{
    int &ri = f();
    int  rj = g();
    f();
    rj = g();
    cout << "ri = " << ri << ", rj = " << rj << endl;

    return 0;
}
```

解：

首先来分析 int &ri = f()。f 函数返回一个引用，实际上就是 f 中静态局部变量 i，而在此之前，i 经过了自增，所以 i 等于 3。独立引用 ri 接受了 f 的返回引用值，实际上就是使 ri 成为 i 的别名。这样一来，此后的 f()调用 i 的值变为 4，那么 ri 的值也就是 4。而对于 rj，在第一次对它的赋值后，rj 等于 3。请注意：函数 g 里的局部变量 i 只是一个自动变量，当 g 返回后，i 变量的生命期也结束了，因此它的值不被保留。所以，当第二次调用 g 对 rj 的赋值后，rj 的值还是等于 3。所以，这个程序的输出应该是：

ri = 4, rj = 3

3. 写一个函数

char * stringcat(const char * s1, const char * s2);

它带有两个串参数，并返回一个串，该串是两个串参数的合并。要求用 new 分配结果串的存储。

解：

```
//辅助函数，求字符串的长度，不包括结尾符号'\0'
//参数：  const char * str，指向源串的指针
//返回值：      int，源串的长度
int stringlen(const char *str)
{
char * p = str; //p 是工作指针

while (*p) p++; //将 p 一直拨到字符串的尾部，此时它指向结尾的'\0'
return p – str; //首尾地址的差值就是串的长度
}

//合并两个字符串，产生新的字符串
//参数：  const char * s1，待合并的第一个字符串
//参数：  const char * s2，待合并的第二个字符串
//返回值：      char *，指向合并后的字符串
char * stringcat(const char * s1, const char * s2)
{
char * str = new char[stringlen(s1) + stringlen(s2) + 1];
                                          //+1 是因为串的结尾有一个'\0'字符
char * p = str; //p 是工作指针

while (*p++ = *s1++); //逐字符拷贝 s1，包括 s1 的结尾'\0'
--p; //因为还没有结束，所以这个'\0'字符不能保留，指针回拨
while (*p++ = *s2++); //逐字符拷贝 s2，包括 s2 的结尾'\0'

return str;
}
```

值得注意的是，stringcat 函数返回的指针指向的存储空间是动态分配的，所以，当这个指针不再使用时，需要显示地使用 delete 运算符来释放它。

4. 写一个程序来求 Fibonacci 数列的前 20 项。Fibonacci 数列的递推定义为：

$$F_n = F_{n-1} + F_{n-2}$$

其中，$F_1 = 1$，$F_2 = 1$，且 $n >= 3$。

解：

递推公式已经给了我们非常明确的算法。在程序中，我们需要设立三个变量 F_n、F_n1、F_n2，分别存储 F_n、F_{n-1}、F_{n-2} 值。每次计算后，将 F_{n-1} 的值赋给 F_{n-2}，将 F_n 的值赋给 F_{n-1}，重复算法，就能求得相应的下一项的 F_n。

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

    int main()
    {
        long Fn, Fn1 = 1, Fn2 = 1;     //Fn1 == F(n-1) , Fn2 == F(n-2)

        cout << "F1 = " << Fn2 << endl;
        cout << "F2 = " << Fn1 << endl;

        for (int i = 3; i <= 20; i++)
        {
            Fn   = Fn1 + Fn2;
            cout << "F" << i << " = " << Fn << endl;
            Fn2 = Fn1;
            Fn1 = Fn;
        }

        return 0;
    }
```

程序的输出结果：

```
F1 = 1
F2 = 1
F3 = 2
F4 = 3
F5 = 5
F6 = 8
F7 = 13
F8 = 21
F9 = 34
F10 = 55
F11 = 89
F12 = 144
F13 = 233
F14 = 377
```

F15 = 610
F16 = 987
F17 = 1597
F18 = 2584
F19 = 4181
F20 = 6765

5. 一家商店卖的某种货物有一包 6 件、9 件和 20 件 3 种包装。这种货物只整包出售，不开零销售。这样一来，客户要求的数目必须是 6、9、20 三个数的组合。比如：客户要买 21 件货物，那么可以买两包 6 件装、一包 9 件装的。很明显，不是所有的数都能由 6、9、20 组合而成，比如 22、23。请编写一个程序来求不能组合的最大数是多少。

解：

这个问题的实质是求解方程

$$6x + 9y + 20z = N \quad （N 是正整数）(*)$$

的自然数解的问题。

解法 1：

一个非常自然而且容易想到的解法就是枚举。将 x、y、z 在一定变化范围内的所有能够组合的数计算出来，然后观察它们的规律，就可以找到答案。根据这个想法，程序设计如下所述。

```
#include <iostream>
#include <iomanip>

using namespace std;

const int X = 10;
const int Y = 6;
const int Z = 3;

int Results[6 * X + 9 * Y + 20 * Z];

int main()
{
    int x, y, z, k;

    for (x = 0; x < X; x++)
        for (y = 0; y < Y; y++)
            for (z = 0; z < Z; z++)
            {
```

```
                k = 6 * x + 9 * y + 20 * z;
                Results[k] = k;
            }

        for (x = 0; x < 6 * X + 9 * Y + 20 * Z; x++)
            cout << setw(4) << Results[x];

        return 0;
    }
```

程序的输出如下：

```
0   0   0   0   0   0   6   0   0   9   0   0  12   0   0  15   0   0  18   0
20  21   0   0  24   0  26  27   0  29  30   0  32  33   0  35  36   0  38  39
40  41  42   0  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79
80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95   0  97  98  99
100 101   0 103 104   0 106 107   0 109 110   0 112 113   0 115   0   0 118 119
0 121   0   0 124   0   0 127   0   0 130   0   0 133   0   0   0   0   0 139
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

观察输出结果，可以发现：从 44 起，以后的数都是连续的；后面出现的 0 是因为覆盖范围不够造成的。因此，可以得出结果：43。

解法 2：

解法 1 虽然简单，但不够严密。因此我们需要更好的解法。

通过观察，可以得出：如果 N 可以被组合，那么 N+6 一定能被组合。故此，我们可以断言：如果 N 能被组合，并且 N+1、N+2、N+3、N+4、N+5 都能被组合，那么从 N 开始，此后所有的数都能被组合。所以，只要我们能找到第一个具有上述性质的 N，那么 $N-1$ 就是本题的答案。

那么该如何找到这样的 N 呢？先来看看 N 的性质。我们可以将方程(*)做如下的变形：

$$6x + 9y = N - 20z$$

$$=> 3(2x + 3y) = N - 20z \ (**)$$

从(**)式中可以得出这样的结论：如果方程有解，那么，

1） 如果 x 和 y 都为 0，那么 N 一定是 20 的倍数；

2） 如果 x 和 y 不全为 0，那么 N 减去 20 的某个倍数后一定是 3 的倍数。令这个差值为 M，由于 x、y 不全为 0，则 M 的最小值应该是 3(2*1 + 3*0) = 6。

根据上面的讨论，可以得出这样简要的求解算法：

1）从某个 N(N=21)开始，试探它和其后连续的 5 个数是否能被组合。第一个满足条件的 N 就是要要找的数，并且 $N-1$ 就是本题的答案。

2） 如果其中一个不能被组合，假设它是 N+i(i=0,1,2,3,4,5)，那么就让 $N = N + i$，

然后重复第一步。

判断 *N* 能否被组合可以通过 *N* 的性质来解决。

下述是程序实现。

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

bool CanCombinated(int n)
{
    if (n % 20 == 0) return true; //20 的倍数一定能被组合，
    //否则
    int nCount = n / 20; //n 能被 20 减的次数
    for (int i = 0; i <= nCount; i++, n -= 20) //每次减去 20
    {
        if (n % 3 == 0 && n >= 6) return true;
    }

    return false; //很遗憾，N 不能被组合
}

int main(void)
{
    bool bCanComb = true;
    int N = 21, i;

    while (true)
    {
    cout << "Testing key number " << N << "..." << endl;

        bCanComb = true;
        for (i = 0; i < 6 && bCanComb; i++)
            bCanComb = CanCombinated(N + i);
        if (bCanComb == true) break; //连续 6 个数都能被组合，得到答案

        N += i;
    }

    cout << "The result is : " << N - 1 << endl;

    return 0;
```

```
}
```

程序的输出结果是：

```
Testing key number 21...
Testing key number 23...
Testing key number 24...
Testing key number 26...
Testing key number 29...
Testing key number 32...
Testing key number 35...
Testing key number 38...
Testing key number 44...
The result is : 43
```

6. 已知一个信号在 x 和 y 方向上的电压变化呈线性关系。现有三组实测电压值：(–1, 6)、(1, 1)和(3, –4)。请根据实测值拟合出该信号的线性方程。

解：

既然是线性关系，那么 x 和 y 的值一定满足方程：

$$y = ax + b$$

其中，a,b 是待定系数。

将已知条件代入（从三组中任选两组），就得到一个线性方程组：

$$6 = a * (-1) + b \Rightarrow -a + b = 6$$

$$1 = a * 1 + b \Rightarrow a + b = 1$$

所以，问题最后转换为对线性方程组求解的问题。线性方程组的求解有很多方法，这里我们用高斯消元法。

在这个问题中，方程中的系数矩阵是：

$$\begin{matrix} -1 & 1 \\ 1 & 1 \end{matrix}$$

它的常量（即初始解）向量是：（6 1）。

下述是程序实现。

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

const int MAXDIM = 10;

int main()
{
    double coef[MAXDIM][MAXDIM+1];
```

```
                                        //存放数组的系数和常量向量，0 行 0 列不用
int n;
double s[MAXDIM];                       //此数组用于存放方程解
int k, i, j;
int flag;

cout << "Please input the dimension of array: (<10)";
cin >> n;

cout << "Please input the coefficient matrix and constant vector:" << endl;
cout << "Format: coef1 coef2 ... coefn cvect" << endl;
for (i = 1; i <= n; i++)
{
    cout << "Line " << i << ':';
    for (j = 1; j <= n + 1; j++)
        cin >> coef[i][j];
}

double temp;
for (k = 1; k <= n - 1; k++)
{
    temp = 0;
    for (i = k; i <= n; i++)
    if (fabs(coef[i][k]) > temp)
    {
        temp = fabs(coef[i][k]);
        flag = i;
    }
    if (flag != k)
    {   //交换行
        for (i = 1; i <= n + 1; i++) coef[0][i] = coef[flag][i];
        for (i = 1; i <= n + 1; i++) coef[flag][i] = coef[k][i];
        for (i = 1; i <= n + 1; i++) coef[k][i] = coef[0][i];
    }
    for (i = k + 1; i <= n; i++)
        for (j = k + 1; j <= n + 1; j++)
            coef[i][j] -= coef[k][j] * coef[i][k] / coef[k][k];
}
```

```
        s[n] = coef[n][n + 1] / coef[n][n];
        for (k = n − 1; k >= 1; k--)
        {
            temp = 0;
            for (j = k + 1; j <= n; j++) temp += coef[k][j] * s[j];
            s[k] = (coef[k][n + 1] - temp) / coef[k][k];
        }

        cout << "The solutions is following:" << endl;
        for (i = 1; i <= n; i++) cout << "s" << i << " = " << s[i] << endl;

        return 0;
    }
```

程序的输出结果是：

```
Please input the dimension of array: (<10)2
Please input the coefficient matrix and constant vector:
Format: coef1 coef2 ... coefn cvect
Line 1:-1 1 6
Line 2:1 1 1
The solutions is following:
s1 = -2.5
s2 = 3.5
```

1.2 第 3 章 类和对象

1. 3.1 节中提出的 Date 结构的操作有很多限制。现在请你将它改写成一个完整的类，同时类的操作要突破那些限制。比如说，AddYear()的参数可以是任意整数值。

解：

一般的日期类型都包含 year/month/day 三个分量，这里我们将它们封装在 Date 类中作为私有数据处理。

为了推算某个日期，我们可以采用这样的算法：

1） 如果是增减年份，就直接在 year 分量上加减。不过需要注意，当原来的 year 是闰年，而加减后的 year 不是闰年并且月份恰恰是 2 月的时候，可能需要调整当月的日值。

2） 如果是增减月份，就先算出要增减的月数要跨过几年和几个月，然后再分别增减 year 和 month。具体的计算方法如下：设增减的月数为 m，那么，要增减的年数和月数分别为

YearSpan = m / 12
MonthSpan = m　% 12

则

year += YearSpan;
month += MonthSpan;

但这可能导致 month<0。因此在这种情况下，year 还要再减 1，而 month 加上 12 就行了。

另外一种情况就是 month 可能超过 12，所以还要再一次调整年和月。

3） 如果增减天数，那么我们可以将目前的日期换算成离 year 年 1 月 0 日一共有多少天，然后加上要增减的天数，最后再将天数切分成几月几日。如果天数为负数，就把这个天数加上当年的总天数（365 或 366），同时年数减 1，直到天数为正后再进行切分。如果天数为正，操作正好相反。

以下是具体的程序：

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

class   Date
{
private:
   int         day, month, year;
   int         GetDayUpperBound();  //获取 month 月的天数上限
   int         GetDaysInYear();
   bool IsLeapYear(int year);
   bool IsLeapYear();

   static int        DayUpperBound[13];

public:
   void InitDate(int d, int m, int y); //初始化
   bool Validate(); //验证日期数据的合法性
   void AddYear(int y);      //如果参数为负数，则表明要计算过去
   void AddMonth(int m);
   void AddDay(int d);
   void Print();   //按年-月-日的格式打印日期
};

int Date::DayUpperBound[] = { -1/*reserved*/,31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
```