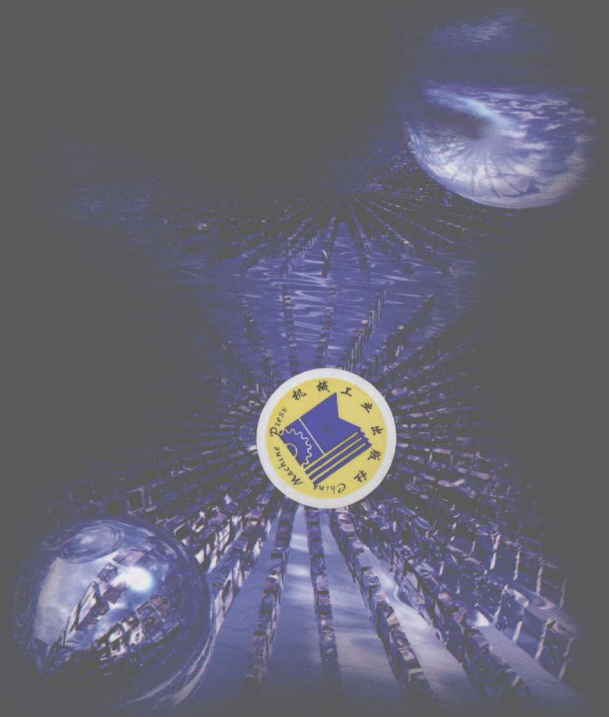
免费提供教学资源

高等院校规划教材
计算机科学与技术系列

面向对象程序设计语言C++习题解答与上机指导

陈文宇　白忠建　戴　波　编著



机械工业出版社
CHINA MACHINE PRESS

高等院校规划教材·计算机科学与技术系列

面向对象程序设计语言 C++ 习题解答与上机指导

陈文宇　白忠建　戴　波　编著

机械工业出版社

本书是《面向对象程序设计语言 C++》（第 2 版）（ISBN 978-7-111-13714-6）的配套指导书。本书包括如下主要内容：主教材的习题参考解答，包括习题的解题分析和相关源代码；C++上机操作指南，通过示例和图表，深入浅出地对 C++常用开发环境（Visual C++ 6.0 和 MinGW）的软件安装及程序的创建、调试排错和运行一一进行了详细介绍；实验指导，包含了 11 个涵盖教材所有章节主要内容的实验，指导学生通过实验编程逐步掌握 C++解决问题的方法和手段，每个实验还给出了参考源代码和运行结果。

本书可作为高等院校本科生的教学参考书和实验指导书，也可供从事计算机软件开发和应用的人员自学和参考。

图书在版编目(CIP)数据

面向对象程序设计语言 C++习题解答与上机指导/陈文宇，白忠建，戴波编著. —北京：机械工业出版社，2009.11
（高等院校规划教材·计算机科学与技术系列）
ISBN 978-7-111-28055-2

Ⅰ. 面… Ⅱ. ①陈…②白…③戴… Ⅲ. C 语言—程序设计—高等学校—教学参考资料 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2009）第 144969 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
责任编辑：陈 皓
责任印制：杨 曦

北京蓝海印刷有限公司印刷

2010 年 1 月第 1 版·第 1 次印刷
184mm×260mm ·14.75 印张·365 千字
0001—3000 册
标准书号：ISBN 978-7-111-28055-2
定价：27.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：(010)88361066	门户网：http://www.cmpbook.com
销 售 一 部：(010)68326294	教材网：http://www.cmpedu.com
销 售 二 部：(010)88379649	**封面无防伪标均为盗版**
读者服务部：(010)68993821	

出 版 说 明

计算机技术的发展极大地促进了现代科学技术的发展，明显地加快了社会发展的进程。因此，各国都非常重视计算机教育。

近年来，随着我国信息化建设的全面推进和高等教育的蓬勃发展，高等院校的计算机教育模式也在不断改革，计算机学科的课程体系和教学内容趋于更加科学和合理，计算机教材建设逐渐成熟。在“十五”期间，机械工业出版社组织出版了大量计算机教材，包括“21 世纪高等院校计算机教材系列”、“21 世纪重点大学规划教材”、“高等院校计算机科学与技术‘十五’规划教材”、“21 世纪高等院校应用型规划教材”等，均取得了可喜成果，其中多个品种的教材被评为国家级、省部级的精品教材。

为了进一步满足计算机教育的需求，机械工业出版社策划开发了“高等院校规划教材”。这套教材是在总结我社以往计算机教材出版经验的基础上策划的，同时借鉴了其他出版社同类教材的优点，对我社已有的计算机教材资源进行整合，旨在大幅提高教材质量。我们邀请多所高校的计算机专家、教师及教务部门针对此次计算机教材建设进行了充分的研讨，达成了许多共识，并由此形成了“高等院校规划教材”的体系架构与编写原则，以保证本套教材与各高等院校的办学层次、学科设置和人才培养模式等相匹配，满足其计算机教学的需要。

本套教材包括计算机科学与技术、软件工程、网络工程、信息管理与信息系统、计算机应用技术以及计算机基础教育等系列。其中，计算机科学与技术系列、软件工程系列、网络工程系列和信息管理与信息系统系列是针对高校相应专业方向的课程设置而组织编写的，体系完整，讲解透彻；计算机应用技术系列是针对计算机应用类课程而组织编写的，着重培养学生利用计算机技术解决实际问题的能力；计算机基础教育系列是为大学公共基础课层面的计算机基础教学而设计的，采用通俗易懂的方法讲解计算机的基础理论、常用技术及应用。

本套教材的内容源自致力于教学与科研一线的骨干教师与资深专家的实践经验和研究成果，融合了先进的教学理念，涵盖了计算机领域的核心理论和最新的应用技术，真正在教材体系、内容和方法上做到了创新。同时本套教材根据实际需要配有电子教案、实验指导或多媒体光盘等教学资源，实现了教材的“立体化”建设。本套教材将随着计算机技术的进步和计算机应用领域的扩展而及时改版，并及时吸纳新兴课程和特色课程的教材。我们将努力把这套教材打造成为国家级或省部级精品教材，为高等院校的计算机教育提供更好的服务。

对于本套教材的组织出版工作，希望计算机教育界的专家和老师能提出宝贵的意见和建议。衷心感谢计算机教育工作者和广大读者的支持与帮助！

机械工业出版社

前　言

C++语言是从C语言发展而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。面向对象语言，如Java、C#等，都与C++有相似的语法结构，如抽象出同类型对象的共性称为类，子类继承父类的属性和行为等。由于C++的突出优点，使其成为目前应用最为广泛的一种面向对象语言，但也被大多数人认为是一种复杂的、难以学习的语言。因此，《面向对象程序设计语言C++》（第2版）（ISBN 978-7-111-13714-6）用大量的例子有序地、深入浅出地讲解了面向对象语言的特点，每一章节都配有丰富的习题供学生练习，帮助学生深入理解面向对象语言和C++的性质。为了帮助学生使用好教材，更好地理解面向对象语言和使用C++开发环境进行程序设计，笔者特编写了《面向对象程序设计语言C++习题解答与上机指导》一书。

本书共分4章，第1章是与教材配套的习题参考解答，给出了问题的详细分析和解题思路，并提供了源代码；第2章是C++上机指南，分别针对最广泛使用的C++集成开发环境Visual C++ 6.0和最接近Win32的编译系统MinGW，从软件的安装步骤，到该环境下程序的创建（编辑源程序），以及编译、链接和运行等各个方面，结合图例进行了详细介绍；第3章是调试和排错，通过各种示例，引出各种初级程序员容易犯的错误，详细介绍了程序调试排错的方法，例如，如何定位错误，如何分析错误原因及解决问题的手段和方法；第4章是实验指导，根据教材每一章的重点内容配备一个相关实验，根据实验题目、目的和要求来指导学生学习掌握相关章节的主要内容，也帮助学生通过实验逐步掌握流程化的C++程序设计技能。另外，每个实验都附有源代码和实验运行结果。所有实验都是在编者积累多年的教学经验基础上精炼而成的，并通过了本科生的教学实践验证，实验效果良好。

本书内容由浅入深，条理清楚，层次分明，叙述详尽。其中的调试与排错是广大C++学习者最难掌握的部分，且相关文献较少，本书的示例涵盖了各种错误和调试方法，对于缺乏经验的C++程序员有着很好的示范作用。本书各个章节提供的大量源代码可以起到很好的示例作用，读者可以登录机工教材网（http://www.cmpedu.com）下载所有的源代码。

本书由电子科技大学计算机学院的陈文宇担任统筹、规划和审校工作，编写工作由电子科技大学计算机学院的白忠建、陈文宇和戴波完成。其中，白忠建编写了第1～3章，陈文宇和戴波编写了第4章。

在此，谨对指导实验的老师和广大参与实验的学生表示衷心的感谢。

由于作者水平有限，书中难免存在疏漏和不足，敬请广大读者批评指正。

编　者

目　录

第1章　习题参考解答

1.1　主教材第2章　C++语法习题参考解答

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：

首先来分析表达式 a+=f(g(a))。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，函数 g 的形式参数是引用，所以函数的形参 i 可以看做是实参 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 的一个复制（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 函数又作用在 b 上，使 b 本身发生了变化（此时，b==2）。再后来执行 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

复制（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 中的静态局部变量 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++);            //逐字符复制 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++);            //逐字符复制 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，需要显式地使用 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 $n \geqslant 3$。

解：

递推公式已经给出了非常明确的算法。在程序中，需要设立 3 个变量 Fn、Fn1、Fn2，分别存储 F_n、F_{n-1}、F_{n-2} 的值。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 这 3 个数的组合。例如，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，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

$6x + 9y + 20z = N$（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：

```
#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。因此，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。可以将方程（*）进行如下的变形：

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

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

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 $3(2\times1 + 3\times0) = 6$。

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。

下面是程序实现：

```
#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 3 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 3 组中任选两组），就得到一个线性方程组：

$$6=a\times(-1)+b=> \quad -a+b=6$$

$$1=a\times 1+b=> \quad a+b=1$$

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，这里用高斯消元法。

在这个问题中，方程中的系数矩阵是：

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

它的常量（即初始解）向量是：（6　1）。

下面是程序实现：

```
#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 主教材第 3 章　类和对象习题参考解答

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。例如，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 这 3 个分量，这里将它们封装在 Date 类中作为私有数据对待。

为了推算某个日期，可以采用以下的算法。

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。

2）如果是增减月份，就要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。具体的计算方法为：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，则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, 30, 31 };

bool Date::IsLeapYear(int year)
{
      return      (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
```