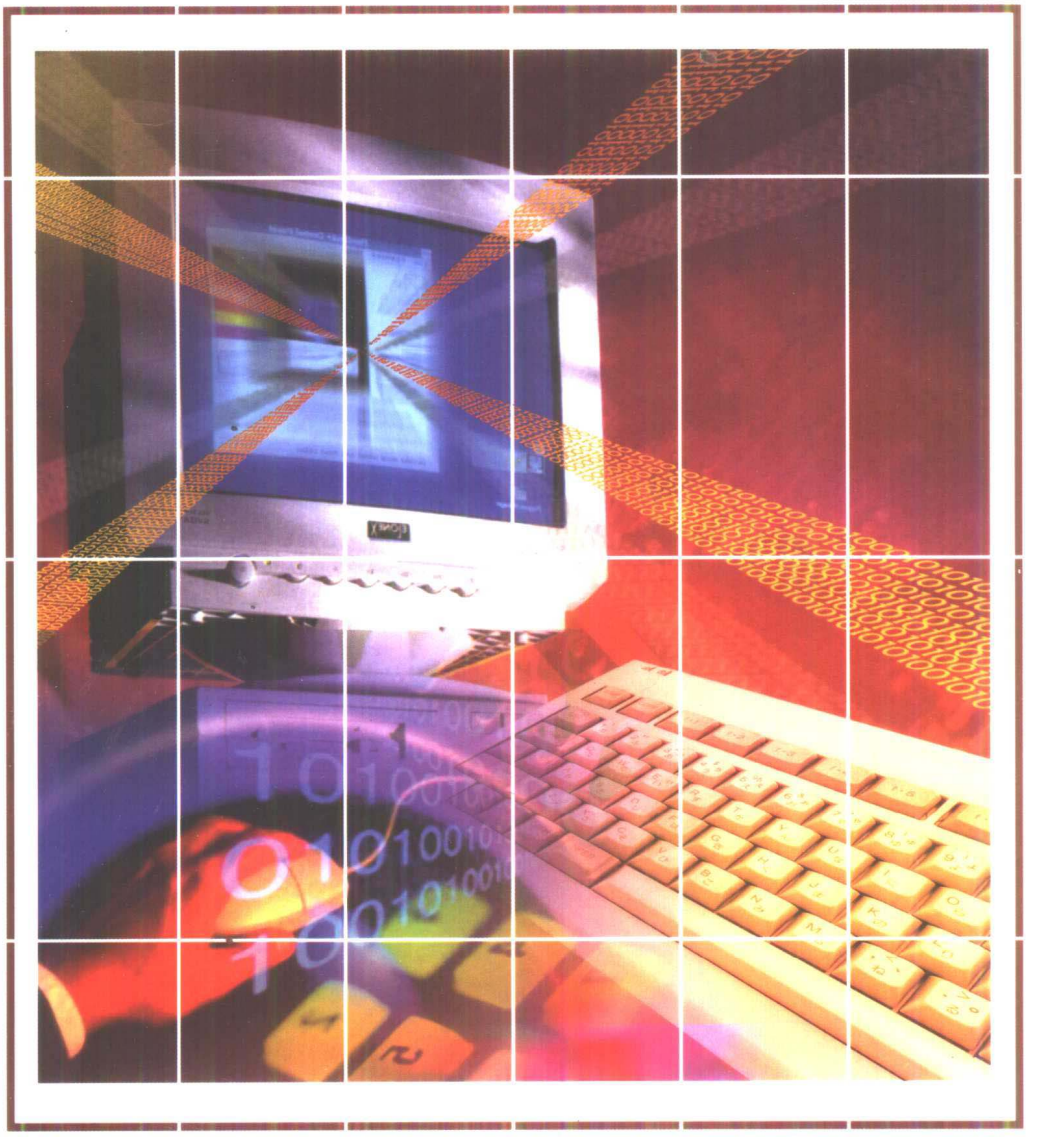
新世纪计算机类本科系列教材



数据结构——使用C++语言

朱战立 编著

西安电子科技大学出版社
http://www.xduph.com

□新世纪计算机类本科系列教材

数 据 结 构

——使用C++语言

朱战立 编著

西安电子科技大学出版社
2001

内 容 简 介

数据结构是计算机等专业必修的核心课程，本书是为高等院校开设该课程精心编著的教材。书中以面向对象观点讨论各种数据结构的设计和应用问题，用C++语言描述和实现了20多个类的设计和典型应用问题的程序设计。本书讨论的典型数据结构问题包括表、堆栈、队列、数组、串、树、二叉树、图、递归程序设计、排序和查找方法。介绍的典型存储结构包括顺序存储结构、链式存储结构、间接地址和仿真指针。还以典型数据结构问题为例子分析了分而治之和回溯法的算法设计方法。

本书既可作为大专院校计算机等专业数据结构课程的教科书，也可作为从事计算机开发和应用的工程技术人员的自学参考书。

图书在版编目(CIP)数据

数据结构：使用C++语言/朱战立编著. —西安：西安电子科技大学出版社，2001.2

新世纪计算机类本科系列教材

ISBN 7-5606-0989-9

Ⅰ.数… Ⅱ.朱… Ⅲ.数据结构-高等学校-教材 Ⅳ.TP311.12

中国版本图书馆CIP数据核字(2001)第00048号

责任编辑 陈宇光 李纪澄

出版发行 西安电子科技大学出版社(西安市太白南路2号)

电 话 (029)8227828 邮 编 710071

http://www.xduph.com E-mail：xdupfxb@pub.xaonline.com

经 销 新华书店

印 刷 陕西画报社印刷厂

版 次 2001年2月第1版 2001年2月第1次印刷

开 本 787毫米×1092毫米 1/16 印张 20.825

字 数 496千字

印 数 1～4 000册

定 价 22.00元

ISBN 7-5606-0989-9/TP·0889

前　言

数据结构是计算机专业、信息管理专业和其他一些与计算机技术关系密切专业的一门必修的核心课程。数据结构课程的任务是讨论现实世界中数据的各种逻辑结构、在计算机中的存储结构以及实现各种操作的算法问题。数据结构课程的目的是使学生掌握如何组织数据、如何存储数据和如何处理数据的基本方法，从而为以后进行软件开发和应用、为进一步学习后续专业课程打下坚实的基础。

计算机软件开发方法是不断发展的，数据结构课程内容也应随着软件开发方法的不断发展而更新。目前，面向对象的软件分析和设计技术已发展成为软件开发的主流方法，因此，用面向对象的方法描述数据结构就成为数据结构课程内容改革的必然。用面向对象的观点讨论数据结构的教材内容在数据逻辑结构、存储结构的描述方面与传统的用面向过程的观点讨论数据结构的教材内容一样，但在其他涉及软件设计方法的内容方面差别却很大。

当要用面向对象的观点描述具体的数据结构问题时，要涉及到选用什么样的面向对象的高级语言问题。目前，C++语言是普遍采用的面向对象程序设计的最底层语言，因此本书采用C++语言描述具体的数据结构问题。本书的所有例子都用 Visual C++ 6.0 调试通过。

近10年来，作者一直讲授数据结构课程，考虑到数据结构课程内容要不断发展更新，曾于1997年编著出版了用C语言描述的数据结构，该教材是当时全国较早推出的同类教材之一。在该教材受到普遍欢迎和作者进一步充实完善的基础上，又于2000年出版了该教材的第2版。作者近年来还一直从事研究生高级数据结构课程的教学，用面向对象的方法描述数据结构是高级数据结构课程教学的主要内容之一。

数据结构教材的编著方法有两种：一种是注重理论基础，用某种类计算机高级语言(如类C语言)描述；另一种是注重工程实用，用某种计算机高级语言直接描述，并给出完整的程序设计例子。作者以前出版的用C语言描述的数据结构教材和本书都采用后一种编著方法。注重工程实用的数据结构教材能通过实际例子帮助学生理解数据结构课程中的各种基本概念，并能培养和训练学生软件开发和设计的实际能力。

本书的主要内容包括：讨论了包括表、堆栈、队列、数组、串、树、二叉树、图、递归程序设计、排序和查找方法等典型数据结构问题，介绍了包括顺序存储结构、链式存储结构、间接地址和仿真指针的典型存储结构，分析了分而治之和回溯法的算法设计方法，给出了20多个类的分析、

设计和应用讨论。

全书共分 10 章。第 1 章简单讨论了 C++面向对象程序设计的要点，这是对学生以前所学 C++面向对象程序设计语言的总结和复习。第 2 章简要介绍了面向对象程序设计的思想和数据结构中的算法性能分析，并在 2.4.4 节给出了算法的符号命名、书写格式和注释格式。第 3 章介绍了顺序存储结构，讨论了顺序表、顺序堆栈、顺序队列和顺序优先级队列类的设计和应用问题。第 4 章介绍了链式存储结构，讨论了单链表、单循环链表、双向循环链表、链式堆栈和链式队列类的设计和应用问题。第 5 章介绍了间接地址存储结构，讨论了数组和串的类设计问题，以及特殊矩阵和稀疏矩阵的压缩存储类设计问题。第 6 章介绍了递归概念和递归算法的设计方法，以实际例子分析了常用的分而治之和回溯法的算法设计方法。第 7 章介绍了仿真指针存储结构，讨论了树和二叉树的各种存储结构，以及树、二叉树、二叉树遍历游标、线索二叉树、堆类的设计和应用问题，并讨论了哈夫曼树的概念和设计问题。第 8 章介绍了图的概念和各种存储结构，讨论了图类的设计问题，以及最小生成树和最短路径的几个常用算法。第 9 章介绍了插入排序、选择排序、交换排序、归并排序和基数排序的一些常用算法。第 10 章介绍了顺序表查找、索引表查找、树表查找和哈希表查找问题。面向对象程序设计方法的主体是类设计，本书从注重工程实用的角度出发，共讨论了 20 多个类的设计和应用问题。

数据结构课程是一门实践性非常强的课程，习题的选择和设计对学生学习和掌握好本门课程十分重要，因此，作者在习题选择和设计上考虑了习题的完整性、系统性和典型性，并把所有习题按类型分成基本概念习题、复杂概念习题、算法设计习题和上机练习习题等四大类。此外，本书的习题数量很丰富，这些习题既是对课程学习的复习和补充，也是考试复习的复习大纲。习题分类设计既方便了学生学习，也方便了教师布置书面作业和上机作业。根据作者经验，使用本教材授课约需 50～70 学时。

尽管作者竭尽全力、非常认真地编写了本书，但由于水平有限，错误和不足之处在所难免，敬请读者批评指正。

朱 战 立

2000 年 10 月

目　录

第 1 章　C++面向对象程序设计要点

本书将以支持面向对象程序设计的C++语言描述数据结构，因此掌握C++编程方法是学习本书的基础。本章概述C++面向对象程序设计的要点。

1.1　函数和函数参数

1.1.1　函数类型

C++中有两种类型函数：常规函数和成员函数。常规函数用于完成一个特定的功能，成员函数用于类方法的定义。无论是常规函数还是成员函数，其定义都包括四个部分：函数名、形式参数表、返回类型和函数体。函数的使用者通过函数名来调用函数，调用过程是把实际参数表传送给形式参数表作为函数的数据来源，然后执行函数体中的语句实现该函数的功能，最后得到的返回值由函数名带回函数的调用者。成员函数是用于类方法定义的，因此，只有该类的对象才能调用其成员函数。

1.1.2　函数名重载

C++允许函数名重载，即允许若干个相同的函数名能在相同作用域内使用。编译系统根据函数引用的参数类型和参数个数判明函数的使用。

例1-1　函数名重载

```
#include <stdio.h>
int max(int a, int b)
{
    if(a >= b) return a;
    else return b;
}
float max(float a, float b, float c)
{
    if(a >= b && a >= c) return a;
    else if(a < b && b > c) return b;
```

```
    else return c;
}
void main()
{
    printf("max(2, 3) = %d\n", max(2, 3));
    printf("max(2.3, 5.6, 3.1) = %f\n", max(2.3, 5.6, 3.1));
}
```

在例 1-1 中，两个 max 函数的返回类型不同，参数个数也不同。函数名重载功能增强了函数设计的通用性，这在类的方法设计中特别有用。

1.1.3 函数参数

C++中函数(包括常规函数和成员函数)参数有四种方式：值参数、常值参数、引用参数和常值引用参数等方式。但常值参数方式因无必要，一般不用。要注意的是，不同方式的参数(不同方式的返回值也类同)在函数的传递调用中系统将视为不同的数据类型，如定义有函数 f1(int x)，在函数 f1( )中又要调用函数 f2( )，如果函数 f2( )的定义是 f2(const int x)，系统将视 int 和 const int 为不同的数据类型，从而告知用户数据类型不匹配。int 和 int& 的情况也一样。注意到这一点将可免去许多程序调试中的问题。

例 1-2 函数参数的值参数、引用参数和常值引用参数三种方式。

```
int Example1(int a, int b, int c)                    //值参数方式
{
    int x, y, z;
    x = a; y = b; z = c;
    a = 2 * a; b = 2 * b; c = 2 * c;                 //函数返回后对应实际参数的值不变
    return (x + y + z)/3;
}
int Example2(int& a, int& b, int& c)                 //引用参数方式
{
    int x, y, z;
    x = a; y = b; z = c;
    a = 2 * a; b = 2 * b; c = 2 * c;                 //函数返回后对应实际参数的值改变
    return (x + y + z)/3;
}
int Example3(const int& a, const int& b, const int& c) //常值引用参数方式
{
    return (a + b + c)/3;                            //不允许改变常值引用参数 a, b, c 的值
}
```

引用参数的方法是在类型名后加符号 &，常值引用参数的方法是在引用参数方法的基础上再在函数定义行的最前边加保留字 const。

对于 Example1 的值参数方式，在运行时，对应的实际参数的值拷贝给形式参数，当函数终止时，形式参数的值不拷贝回实际参数。因此，值参数方式的函数调用不会改变对应

形式参数的实际参数的值。

对于 Example2 的引用参数方式，由于引用变量是实际变量的别名，所以函数运行时实际参数的值放到了对应的形式参数的位置，当函数运行时任何对形式参数的改变也就是对实际参数的改变。当函数需要把形式参数的改变值带回到实际参数时引用参数方式是必须的。

对于 Example3 的常值引用参数方式，保留字 const 指明了引用参数的值在函数运行时不会也不能改变。const 保证只能对其值初始化一次，即实际参数的调用前赋值。

1.1.4 成员函数返回值

和函数参数的方式类同，成员函数返回值也有值方式、常值方式、引用方式和常值引用方式等四种方式。

当成员函数的返回值为值方式时，允许改变该对象的私有成员数据。注意，成员函数的返回值为无 const 标识符的任何数据类型(包括 void 类型)时即为值方式。

当成员函数返回值为常值方式时，const 标识符使该对象的私有成员不能被改变。

当成员函数返回值为引用方式时，由于引用型变量不是单独定义的，所以该成员函数返回值是一个已存在变量(或对象)的别名。当该成员函数被重新赋值时，则其对应的变量(或对象)值将改变。

当成员函数返回值为常值引用方式时，其返回值和引用方式的成员函数返回值类同，只是该成员函数返回值不能改变。当成员函数调用后，引用方式的成员函数名所对应的变量(或对象)值不能改变时，应加 const 标识符限定为常值引用方式。

当成员函数返回值为常值方式或常值引用方式时，const 标识符一般放在最后。

下面的例 1－3 仅说明成员函数返回值的值方式和常值方式，其余的情况将在后面章节的例子中说明。

例 1－3 成员函数返回值的值方式和常值方式

```
#include <iostream.h>

class Temperature
{
    private:
        float highTemp                          //最高温度
        float lowTemp;                          //最低温度
    public:
        Temperature (float h, float l){highTemp = h;lowTemp =l;};
        void UpdateTemp (float temp);           //成员函数返回值为值方式
        float GetHighTemp (void) const;         //成员函数返回值为常值方式
        float GetLowTemp (void) const;          //成员函数返回值为常值方式
};

void Temperature::UpdateTemp (float temp)
{
```

```
    if(temp > highTemp) highTemp = temp;    //允许更改私有数据
    if(temp < lowTemp) lowTemp = temp;      //允许更改私有数据
}

float Temperature::GetHighTemp (void) const
{
    return highTemp;                        //不允许更改私有数据
}

float Temperature::GetLowTemp (void) const
{
    return lowTemp;                         //不允许更改私有数据
}
```

1.2 输 入 输 出

C++提供了一个层次结构的I/O流类来支持输入输出。I/O流类的类层次结构关系如图1-1所示。流是一个非常形象的术语，每一个流类指定了一类文件和这类文件数据流动的方向。当我们定义一个具体的流类对象时，就确切地指定了这个文件的存储区和由系统设定的内存缓冲区，以及这个文件在其存储区和内存缓冲区之间的数据流动方向。

ios类是基类，它提供了格式、状态信息和错误检测。类ios中没有定义实际的输入和输出操作。实际操作是由派生类istream、ostream等提供的。

两种基本的输入输出方式是：键盘屏幕(默认)输入输出和文件输入输出。

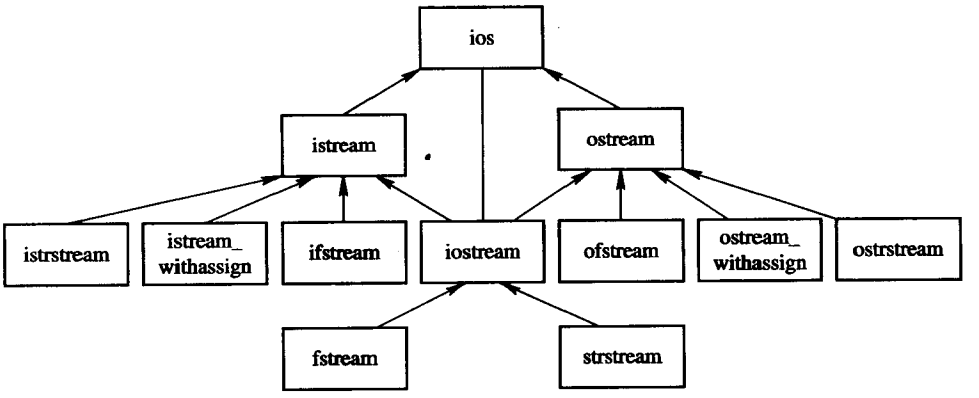


图1-1 I/O流类的类层次

1.2.1 键盘屏幕输入输出

iostream流类中定义了键盘输入类、屏幕输出类和错误信息输出类。当iostream流类

包含在应用程序中时，cin、cout 和 cerr 分别为键盘输入类、屏幕输出类和错误信息输出类的系统默认对象。cin 对象键盘输入的操作符为>>，cout 和 cerr 对象屏幕输出的操作符为<<。

```
#include <iostream.h>

void main(void)
{
    char c;
    int n;
    cout << "输入 n = ";
    cin >> n;
    cout<< "n = " << n << endl;
}
```

1.2.2 文件输入输出

调用打开文件函数时，打开文件函数的参数包括文件名和数据流动的方向，打开文件函数返回文件的开始地址。系统在存储文件时，在其末尾添加有文件结束标记，这样打开文件函数就具体地指定了一个文件在内存中的起始位置、结束位置和数据流动的方向。

fstream.h 流类中定义了文件输入类和文件输出类。文件输入类的类名为 ifstream，文件输出类的类名为 ofstream。

```
#include <fstream.h>
#include <iostream.h>
#include <stdlib.h>                          //包含 exit( )函数

void main(void)
{
    ifstream fin;                            //定义 fin 为 ifstream 类的对象
    ofstream fout;                           //定义 fout 为 ofstream 类的对象

    fout.open("exam.dat", ios::out);         //建立输出文件 exam.dat

    //输出字符串"wang"和数字 50 到输出文件 exam.dat 中(可打开该文件查看)
    char name[] = "wang", name2[10];
    int age = 50, age2;
    fout << name << endl;
    fout << age << endl;

    fin.open("exam.dat", ios::in | ios::nocreate);     //打开输入文件 exam.dat
    if(! fin)
    {
```

```
        cerr << "不能打开文件 exam.dat" << endl;
        exit(1);
    }
    char c;
    //从输入文件 exam.dat 中输入字符串"wang"和数字 50 到 name2 和 age2 中
    fin >> name2 >> c >> age2;

    //屏幕输出查看
    cout << "name2 = " << name2 << endl;
    cout << "age2 = " << age << endl;
}
```

1.3 类

当开发的应用程序的数据类型不是 int、float、char 等系统支持的基本数据类型时，开发者就需定义自己的数据类型。C++中定义自己的数据类型的面向对象方法是使用类。类是C++支持面向对象程序设计的基础。C++中定义类使用标识符 class，类定义包括类中数据的定义和成员函数的定义。

例 1-4 设计一个人民币类，其方法包括设置、加(+)、加等于(+=)，以及输出流操作<<，并用一个主函数验证。

```
#include <iostream.h>
#include <stdlib.h>
enum sign{minus, plus};          //正负数的符号标识

//类 Money 定义部分
class Money                      //类 Money 定义
{
    friend ostream& operator<<(ostream& out, const Money& x) ; //友元
    private:                                                   //私有部分
        long amount;      //以分为单位的钱数，amount 为类 Money 的数据部分
    public:                                                    //共有部分
        //构造函数
        Money(sign s = plus, unsigned long d = 0, unsigned long c = 0);
        //析构函数
        ~Money( ){};

        //共有成员函数
        bool Set(sign s, unsigned long d, unsigned long c);
        Money operator+(const Money& x) const;
        Money& operator+=(const Money& x)
        {amount += x.amount; return *this;}      //this 是默认的当前对象指针
```

```
};
//类 Money 实现部分
Money::Money(sign s, unsigned long d, unsigned long c)
//构造函数。s 为符号，d 为元值，c 为分值
{
    if(c > 99)
    {
        cerr << "分必须小于等于 99" << endl;
        exit(1);
    }
    amount = d * 100 + c;
    if(s == minus) amount = -amount;
}

bool Money::Set(sign s, unsigned long d, unsigned long c)
//设置函数。s 为符号，d 为元值，c 为分值
{
    if(c > 99)
    {
        cerr << "分必须小于等于 99" << endl;
        exit(1);
    }
    amount = d * 100 + c;
    if(s == minus) amount = -amount;
    return true;
}

Money Money::operator+(const Money& x) const      //操作符+重载
{
    Money y;
    y.amount = amount + x.amount;
    return y;
}

ostream& operator<<(ostream& out, const Money& x)
//流类的操作符重载。前边不再需标识符 friend
{
    Money temp = x;                          //注意临时对象 temp 绝对不能定义为引用型
    if(x.amount < 0 )
    {
        out << '-';
        temp = -temp;                        //temp 为 x 的绝对值
    }
```

```
    long d = temp / 100;                    //d 为 amount 的元值
    out << '¥' << d << '.';
    int c = temp - d * 100;                 //c 为 amount 的分值
    if(c < 10) out << '0';                  //分值小于 10 时在十位上添 0
    out << c;
    return out;
}

//主函数部分
void main(void)
{
    Money g, h(plus, 3, 30), hg;
    g.Set(minus, 2, 25);
    hg = h + g;                             //h 和 g 的值均不变
    g += h;                                 //g 的值被改变
    cout << "h + g = " << hg <<endl;
    cout << "g = " << g << endl;
}
```

程序运行输出：

```
h + g = ¥1.05
g = ¥1.05
```

下面我们对 C++类基本要素的分析讨论将以例 1-4 为实例。

1.3.1 类的存取权限

类的数据和成员函数的存取权限分为三种：私有(private)、公有(public)和保护(protected)。

在 private 域中，声明的数据和成员函数构成类的私有部分，私有部分中的数据和成员函数只能由该类对象的成员函数，以及被声明为友元的函数或声明为友元的类的对象的成员函数访问。

在 public 域中，一般仅声明类的成员函数(有些情况下，也在此域中声明类的数据)。在 public 域中，声明的类的数据和成员函数构成类的公有部分；公有部分中的数据和成员函数既允许该类对象的成员函数访问，也允许程序中其他函数或其他类的对象的成员函数访问。因此，一个类的公有部分就构成了这个类的操作界面。外部函数和别的对象通过操作界面对类中的对象进行操作。这就和基本数据类型中 int 类数据均通过加(+)、减(-)、乘(*)、除(/)等操作界面来对该数据类型中所有数据进行操作一样。

在 protected 域中，声明的数据和成员函数与在 private 域中声明的数据和成员函数一起构成类的私有部分，但在 protected 域中声明的数据和成员函数允许该类的派生类访问。

1.3.2 构造函数和析构函数

构造函数是一种特殊的成员函数。构造函数是用来在内存中建立类的具体对象(即在

内存中为该对象分配适当的空间)并对其进行初始化赋值的成员函数。构造函数的名字必须和类的名字相同，构造函数没有返回值。构造函数允许参数有默认值，当构造函数有默认值时，若定义该类的对象时没有给出初始化值则按默认值处理。例 1－4 中对象 h 有初始化值(plus，3，30)，对象 g 和 hg 没有初始化值，因此对象 g 和 hg 的初始化值为默认值(plus，0，0)。一个类允许有多个构造函数。当一个类有多个构造函数时，系统将根据定义对象时的参数类型和参数个数，选择恰当的构造函数来建立对象和对该对象进行初始化。

析构函数是一种当对象被取消时才被自动调用的特殊函数。当一个对象被取消时，析构函数提供了自动释放由构造函数在内存中建立的该类对象的内存区。析构函数的名字是在类的名字前面加上前缀“～”，析构函数没有返回值。由于在一般情况下建立的对象当被取消时，系统都能自动识别出所占用的相应的内存区，所以一般情况下析构函数的函数体为空。函数体为空的析构函数可省略。但当对象被取消时系统不能自动识别出所占用的内存区的类，析构函数的函数体就不能为空，此时析构函数就要负责释放对象所占用的内存区。每个类只能有一个析构函数。

1.3.3 操作符重载

当所设计类的某个成员函数功能和人们熟悉的某个有固定操作符的函数功能类同时，可以用该操作符替换该成员函数名，这称作操作符重载。操作符重载时，操作符前使用标识符 operator。对用户来说，操作符重载使类的成员函数的调用更加方便。如例 1－4 中的加(＋)和加等于(＋＝)成员函数通过操作符重载方便了用户的调用。也就是说，一个类成员函数的操作符重载就是用一个有固定含义的操作符定义和实现该类的相应成员函数。当调用时该操作符可以看成是该操作符左边对象的成员函数。如例 1－4 中的

```
g ＋＝ h
```

g 可看作是加等于(＋＝)成员函数的对象，h 可看作是加等于(＋＝)成员函数的参数。

操作符重载不仅可以更方便地实现相应成员函数的功能，重载的操作符还具有相应的运算优先级和满足结合律，这是类的成员函数所没有的功能。

成员函数的操作符重载时必须满足：

(1) 操作符必须满足运算优先级和结合律，以及参数满足相应的操作符对操作数个数的要求，如二元操作加(＋)必须有也只能有一个参数作为加数。

(2) C++中的所有操作符基本上都可被重载，但逗号操作(，)和条件表达式操作(?:)不允许重载。

(3) 操作符重载时不允许参数有默认赋值。

(4) 当重载的操作符左边的对象属类 B，右边的对象属类 A 时，该重载的操作符应定义为类 A 的友元；否则类 B 的对象不能操作类 A 对象中的数据。

操作符重载不仅可以定义为成员函数，也可以定义为一般函数。当类的定义和实现已经编译完成，而在应用时具体定义的数据类型不支持类的定义和实现中使用的运算符时，定义为一般函数的操作符重载可不改变原先已编译完成的类的定义和实现。一般函数的二元操作符重载需有两个参数，这一点和成员函数的操作符重载不同。3.5.2 节的小于(＜)操作符的重载就是一个一般函数的例子，此时，原先已编译完成的类的定义和实现不用改动。