

全国高职高专规划教材

面向对象的程序 设计C++

Object-Oriented
Programming with C++

刘加海 主 编
杨昱昂 罗晓芳 副主编



科学出版社

www.sciencep.com



全国高职高专规划教材

面向对象的程序设计 C++

刘加海 主 编

杨昱曷 罗晓芳 副主编

科 学 出 版 社

北 京

内 容 简 介

本书是 C++ 程序设计的基础教材, 全书共 7 章。主要内容涉及到 C++ 的基本概念、类与对象、继承、运算符重载、虚拟函数与多态性、模板和异常处理、I/O 流与文件。本书内容通俗易懂、言简意赅、重点突出。内容的安排循序渐进、深入浅出, 以具体实例来分析和阐明 C++ 语言中的概念与原理。

与本书配套的《面向对象的程序设计 C++ 实训教程》, 通过大量的实训案例示范、模仿, 使得学生在短时间内掌握 C++ 程序设计的原理及概念, 并编写高质量的 C++ 源程序。

本书适合作为高等院校及相关专业的本科、专科、高职学生学习 C++ 程序设计的教材以及相应的学习参考书。

图书在版编目(CIP)数据

面向对象的程序设计 C++/刘加海主编; 杨昱晃, 罗晓芳副主编. —北京: 科学出版社, 2003

(全国高职高专规划教材)

ISBN 7-03-011987-8

I. 面... II. ①刘...②杨...③罗... III. C 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字 (2003) 第 065816 号

策划编辑: 李振格/责任编辑: 丁 波

责任印制: 吕春珉/封面设计: 东方人华平面设计部

科 学 出 版 社 出 版

北京东黄城根北街16号

邮政编码: 100717

<http://www.sciencep.com>

新 蕾 印 刷 厂 印 刷

科学出版社发行 各地新华书店经销

*

2003 年 8 月 第 一 版 开本: 787×1092 1/16

2003 年 8 月 第一次印刷 印张: 11

印数: 1—5 000 字数: 243 000

定价: 15.00 元

(如有印装质量问题, 我社负责调换(路通))

全国高职高专规划教材编委会名单

主 任 俞瑞钊

副主任 陈庆章 蒋联海 周必水 刘加海

委 员 （以姓氏笔画为序）

王 雷 王筱慧 方 程 方锦明 卢菊洪 代绍庆

吕何新 朱 炜 刘向荣 江爱民 江锦祥 孙光弟

李天真 李永平 李良财 李明钧 李益明 余根墀

汪志达 沈凤池 沈安衢 张 元 张学辉 张锦祥

张德发 陈月波 陈晓燕 邵应珍 范剑波 欧阳江林

周国民 周建阳 赵小明 胡海影 秦学礼 徐文杰

凌 彦 曹哲新 戚海燕 龚祥国 章剑林 蒋黎红

董方武 鲁俊生 谢 川 谢晓飞 楼 丰 楼程伟

鞠洪尧

秘书长 熊盛新

本书编写人员名单

主 编 刘加海

副主编 杨昱曷 罗晓芳

撰稿人 应玉龙 吴 俊

前 言

面向对象的程序设计 C++ 是一门非常重要的计算机专业的专业基础课, 学习 C++ 不仅是深入理解面向对象程序设计的一些基本概念, 而且非常有助于进一步学习其他的计算机语言, 如 Visual C++ 等。

在 C++ 程序设计这门课程中, 有关面向对象的概念, 构造函数, 虚基类与多义性, 虚函数与多态性是本课程的重点及难点, 而学生在学习过程中往往感到难以掌握, 本书在编写过程中从实际出发, 兼顾最基本的理论知识, 遵循深入浅出、通俗易懂的原则, 概念清晰、逻辑性强, 力求用大量的例子来阐明本课程的重点与难点, 极大地减轻了读者学习 C++ 的困难, 本书不仅适合于高职计算机专业的教材, 还适合作为计算机专业的本科生或夜大、电大等计算机专业的入门教材。

本书内容共分为 7 章, 第 1 章为 C++ 的基本概念, 第 2 章为类与对象, 第 3 章为继承, 第 4 章为运算符重载, 第 5 章为虚拟函数与多态性, 第 6 章为模板和异常处理, 第 7 章为 I/O 流与文件。

本书由刘加海博士担任主编, 杨昱昶、罗晓芳担任副主编, 第 1 章由罗晓芳、刘加海编写, 第 2 章和第 5 章由刘加海编写, 第 3 章由杨昱昶编写, 第 4 章由应玉龙编写, 第 6 章由杨学明编写, 第 7 章由吴俊编写, 全书由刘加海统稿。

由于编者水平有限, 书中难免有疏漏和不妥之处, 敬请读者批评指正。

编 者

2003 年 6 月

目 录

第 1 章 C++的基本概念	1
1.1 引言	1
1.2 C++的单行注释	1
1.3 C++的输入 / 输出流	2
1.4 变量声明的位置	3
1.5 内联函数	4
1.6 默认函数参数	6
1.7 引用参数	7
1.7.1 独立引用	8
1.7.2 引用参数	9
1.7.3 返回引用	10
1.8 const 限定符	11
1.8.1 const 限定符的声明格式	11
1.8.2 带有指针的 const 限定符	11
1.9 域分辨操作符 ::	12
1.10 运算符 new 和 delete	14
1.11 函数重载	16
习题	17
第 2 章 类与对象	20
2.1 类的基本概念	20
2.1.1 类的定义	21
2.1.2 类成员函数的定义方法	23
2.1.3 类对象的定义及引用方法	24
2.2 数据封装	28
2.2.1 类的私有成员	28
2.2.2 类的公有成员	29
2.3 类的 inline 成员函数	29
2.4 类构造函数	29
2.4.1 构造函数的特点	30
2.4.2 构造函数的应用	31
2.5 拷贝构造函数	34
2.6 类的析构函数	36

2.6.1	析构函数的特点	37
2.6.2	析构函数调用顺序举例	37
2.7	类静态成员	39
2.7.1	类的静态数据成员	39
2.7.2	类的静态成员函数	40
2.8	类的友元	42
2.8.1	友元函数	43
2.8.2	友元类	45
2.9	容器类	48
2.10	隐式指针 this	49
2.11	类与结构	52
2.12	类对象与指针	54
2.12.1	指向类对象的指针	54
2.12.2	指向类成员的指针	56
	习题	59
第 3 章	继承	69
3.1	继承与派生	69
3.2	单继承	70
3.2.1	单继承	70
3.2.2	访问基类成员	71
3.2.3	继承的类型	72
3.3	多继承	75
3.4	多继承中派生类的构造函数与析构函数	76
3.5	多继承中的二义性问题	77
3.6	虚基类	78
	习题	80
第 4 章	运算符重载	86
4.1	运算符重载的基本概念	86
4.1.1	C++中可重载的运算符	86
4.1.2	运算符重载的定义形式	87
4.2	成员函数重载运算符	87
4.3	友元函数重载运算符	90
4.4	一元运算符++、--的重载	92
4.5	赋值运算符的重载	95
4.6	二元运算符的重载	96
4.7	重载运算符（）	100
	习题	101
第 5 章	虚拟函数与多态性	105
5.1	静态联编与动态联编	106

5.2 虚拟函数.....	109
5.2.1 虚拟函数的定义.....	110
5.2.2 虚拟函数的调用.....	110
5.3 构造函数和析构函数对虚函数的调用.....	114
5.4 多重继承与虚函数.....	115
5.5 虚拟函数的数据封装.....	116
5.6 纯虚函数与抽象类.....	118
5.6.1 纯虚函数.....	118
5.6.2 抽象类.....	118
5.7 虚拟函数的继承.....	121
5.8 派生类直接调用基类中的虚拟函数.....	122
习题.....	123
第6章 模板和异常处理.....	132
6.1 模板.....	132
6.1.1 函数模板.....	132
6.1.2 类模板.....	136
6.2 异常处理.....	138
6.2.1 异常处理的基本思想.....	138
6.2.2 异常处理的实现.....	139
6.2.3 异常生命周期.....	140
6.2.4 异常规格说明.....	141
6.2.5 异常处理中的构造与析构.....	142
习题.....	143
第7章 I/O流与文件.....	144
7.1 C++流的概念.....	144
7.2 用 ios 类的成员函数实现格式化输入与输出.....	145
7.3 用操纵符实现格式化输入与输出.....	148
7.4 文件的操作.....	151
7.4.1 文件的操作过程.....	151
7.4.2 文件的打开方式.....	151
7.4.3 文件的操作方式.....	152
7.4.4 文本文件应用举例.....	153
7.4.5 二进制文件的操作.....	155
7.4.6 文件的随机读/写.....	156
习题.....	158
主要参考文献.....	164

第 1 章 C++的基本概念

本章要点

- C++中的输入、输出，变量的声明位置
- 内联函数及默认参数的函数
- 引用的概念
- const 限定符、new、delete 运算符的使用
- 函数重载

本章难点

- 内联函数的概念
- 默认参数的函数
- 返回引用的概念
- 带有指针的 const 限定符
- new 运算符的使用

1.1 引言

亲爱的读者，当您迈入 C++ 世界的时候，您对 C 语言一定有了很深的了解，您可知道 C++ 增强了 C 的许多特点，C++ 是 C 的超集，是面向对象的程序设计，C++ 程序设计方法对 C 语言的一些原有内容有了很大程度的增强和改进。本章所讨论的这部分内容，对于我们这些初步接触 C++ 语言的人来说是非常重要的。

欢迎迈入 C++ 的世界！

1.2 C++的单行注释

程序员给程序加上适当的注释能有效提高程序的可读性，对以后的程序维护也会有很大的帮助。尤其是对大型的程序来讲，好的注释更是必不可少的。C++ 编译器支持下面两种形式的注释：

(1) C 语言中经常使用的以 “/*” 开始，以 “*/” 结束的注释形式，这在 C++ 中同样适用。

(2) 以“//”开始的单行注释形式。很多 C++ 的程序员都比较喜欢这一形式的注释。在此种注释中，编译器一遇到“//”就会认为从它之后直到本行尾的所有内容都认为是注释，通常把这种形式的注释叫做单行注释。

例 1.1 C++ 程序的两种注释的应用。

```
/*
本例将给出C++程序的
两种注释方法
*/
#include<stdio.h>
#include<iostream.h>
void main()
{
    printf("你好 C++! \n");           // 输出: 你好C++!
    cout<<"这是面向对象的程序设计"<<endl; // 输出: 这是面向
对象的程序设计 */
}
```

由此可见，两种形式的注释各有千秋。/* */: 一般用于多行的注释，//: 用于单行的注释，当然希望大家最好使用 C++ 风格的注解符号“//”。

1.3 C++ 的输入 / 输出流

在 C++ 程序设计中，除了允许使用标准 C 语言函数库 `stdio.h` 提供的输入输出函数外，还可以使用自己定义的输入输出系统，即标准输出流 `cout` 和标准输入流 `cin`，其定义在标准头文件 `iostream.h` 中找到。

标准输出流 `cout` 与流插入运算符 `<<` 联用，可以代替 C 语言中的 `printf` 函数，并且不需要任何格式，例如：

```
cout<<"I like C++!";
```

表示把字符串“I like C++!”插入输出流 `cout` 中。

```
cout<<x;
```

表示把变量 `x` 的值插入输出流 `cout` 中。

标准输入流 `cin` 与流提取运算符 `>>` 联用，可以代替 C 语言中的 `scanf` 函数，也不需要任何格式，如：

```
cin>>x;
```

表示从输入流 `cin` 中提取 `x` 的值。

例 1.2 使用 `cout`、`cin` 输出、输入数据。

```
#include<iostream.h>
void main()
{
    cout<<"您好!愿您喜欢C++的输入输出。";           //表示输出一个字符串
}
```

```
cout<<2003;                //打印一个整数
cout<<"\n";                //换行
cout<<20.1;                //打印一个实数
cout<<endl;                //换行
cout<<"I am "<<20<<"years old student. "; //连续打印
char name[30];              //变量的声明位置与C有什么区别吗?
int age;
cout<<"please give your name :";
cin>>name;                  //表示键盘输入字符串到变量name中
cout<<"please tell me how old are you? ";
cin>>age;                   //表示键盘输入整型数到变量age中
cout<<"Your name is "<<name<<endl;
cout<<"you are "<<age<<"years old. ";
}
```

下面总结一下 cout、cin 的用法。

(1) cout 是标准输出流，代表标准输出设备，一般指的是屏幕。它把输出运算符右侧的变量或常量的内容输出到屏幕上。

(2) 用 cout 和<<可以输出包含字符串在内的任何基本数据类型。

(3) 在输出语句中，可以通过操纵算子 endl 或换行符 ‘\n’ 来回车换行。

(4) 在一个 cout 语句中，可以连续使用多个输出运算符<<。标准输入 cin：一般和输入运算符>>合用，用来输入变量的值。

(5) cin 是标准输入，代表标准输入设备，一般指的是键盘。它接收用户从标准输入设备上的输入，并把接收到的数据赋给输入运算符右边的变量。

(6) 用 cin 和>>可以为包含字符串在内的任何基本数据类型的变量提供输入。

(7) 在一个 cin 语句中，可以连续使用多个输入运算符>>，例如：

```
cin>>a>>b;
```

(8) 用输入运算符>>输入信息时，不要在变量名前加上地址运算符&。

(9) 输入运算符>>对字符串的处理方式与 scanf() 函数中%s 声明符的方式相同，即当遇到第一个空格时，就停止输入。例如，执行上面程序时，如果输入名字：liu jia，当 C++遇到 liu 和 jia 之间的空格时，就认为输入完毕。所以，name 的值只会是 liu，这一点请注意。

1.4 变量声明的位置

在例 1.2 中，我们可以看到可在程序代码中的任何位置去声明，这不仅能增强程序的可读性，而且也不必在编写某一程序块的开始时就考虑要用到哪些变量。

一般来说，在某一程序块中自始至终都要使用的数据，我们最好把它们集中在程序块的开始处去声明，而对于那些只在小范围内临时使用的变量，例如，某一循环语句的循环控制变量，可以在使用它们的地方就近声明，这样便于程序阅读。

例 1.3 C++变量声明的一个函数例子。

```
#include<iostream.h>
void func(int k)
{
    int total;
    total=0;
    for(int i=0;i<k;i++)           //在此声明变量i有助于理解i在程序中的作用
    {
        total=total+i;
    }
    cout<<"total= "<<total<<endl;
}
void main()
{
    int x;
    cin>>x;
    func(x);
}
```

在进行变量声明的时候，我们还应该注意到以下两点：

(1) 注意变量的使用范围（即变量的作用域）。变量的作用域是从该变量被声明的地方起，直到此声明所在的程序块结束为止，包括其中的子程序块。例如，在例 1.3 中，变量 `total` 的作用域是从定义它的语句开始直到结束函数定义的花括号为止，而变量 `i` 的作用域仅在整個 `for` 循环语句。

(2) 最好在靠近变量使用的地方声明变量，否则，别人会很难读懂你的程序。

1.5 内联函数

引入内联函数的原因及目的是为了解决程序中函数调用的效率问题。它的引入使得编程者只需关心函数的功能和使用方法，而不必关心函数功能的具体实现。当调用一个函数时，参数要装入堆栈中，各个寄存器的内容和状态都需要保存。当函数返回时，还要恢复它们的内容和状态。所以函数的调用需要一定的开销。而使用内联函数时，函数的调用是进行代码的扩展，而不是简单的函数调用。这将提高程序的运行效率，所以把那些使用频繁的函数声明成内联函数是非常有用的。

1. 内联函数的声明方法

声明内联函数的方法是在函数定义的开头前加上关键字 `inline` 即可，代码的写法与一般函数一样。例如：

```
inline int max(int x, int y)
{
    return x>y?x:y;
}
```

其中, `inline` 是关键字, 此时声明 `max(int, int)` 是一个内联函数。

2. 内联函数的特点

编译器将在内联函数被调用的每个地方都插入它的一份拷贝, 而不是编译为一个单独的可调用的代码。这样可以减少调用函数所需要的时间开销, 从而使程序更高效地运行。当然, 这样做的结果会使程序代码变得更长。

从运行的效率来看, 内联函数与宏定义类似, 然而内联函数可以被编译程序所识别, 而宏则是通过简单的正文替换来实现的。内联函数的优点如下。

- 编译程序可以对其参数作类型检查。
- 内联函数的行为与普通的函数一样, 没有宏定义带来的那些副作用。

请分析以下列程序。

例 1.4 内联函数的例子。

```
#include<iostream.h>
inline doub(int x)
{
    return(x*x);
}
void main()
{
    for (int i=1;i<=5;i++)
    {
        cout<<i<<" double is "<<doub(i)<<endl;
    }
    cout<<"1+2 double is "<<doub(1+2);
}
```

程序的运行结果为:

```
1 double is 1
2 double is 4
3 double is 9
4 double is 16
5 double is 25
1+2 double is 9
```

请思考如下程序:

```
#include<iostream.h>
#define doub(x) (x*x)
void main()
{
    for (int i=1;i<=5;i++)
    {
        cout<<i<<" double is "<<doub(i)<<endl;
    }
    cout<<"1+2 double is "<<doub(1+2)<<endl;
}
```

在使用内联函数时，应注意以下几点：

(1) 内联函数只是对小函数有作用。如果函数有许多行，可能就起不到内联函数的作用。

(2) 内联函数不能包含任何的静态变量，不能使用任何循环语句、switch 语句、goto 语句，不能递归，不能有数组。

(3) 由于内联函数的调用是进行代码的扩展，这必将造成重复代码的生成，而使程序过长。所以最好对那些小的函数使用内联函数。

1.6 默认函数参数

函数调用通常会有参数传递，请看下面的函数例子。

```
int add(int x,int y)
{
    int z;
    z=x+y;
    return z;
}
```

因而，函数在调用时需要有参数，如下面的函数调用：

```
add(a,b);
```

而如果用 add () 的调用方式显然是不正确的。因为我们并没有给出调用此函数时的参数值。但是，我们能不能给 add (int, int) 函数一个隐含的参数值呢？也就是说当你调用此函数而没有给出参数值时，函数会自动把这这个隐含值传入函数。这一点在 C++中是可以做到的，只要我们给出如下方式的函数原型：

```
int add(int x=5, int y=10)
{
    int z;
    z=x+y;
    return z;
}
```

在调用 add () 函数时，既可以给出其参数值，也可以不给出参数值。当你没有给出参数值时，编译器会自动为参数传递一个缺省值 x 为 5, y 为 10，这种函数的参数就称为默认的函数参数。此种功能会给我们的程序设计带来很大的灵活性，但在使用默认的函数参数时，应该注意下面的几个问题：

如果对函数声明中给出了默认的函数参数时，当函数调用时，如果省略参数，默认的参数会自动传递给被调用函数。

例 1.5 默认参数的例子。

```
#include<iostream.h>
void display(int x=1000)
{
```

```
    cout<<x<<endl;
}
void main()
{
    display();    //函数在调用时可以缺省参数,这时x的默认参数为1000,程序输出为1000
}
```

函数在调用中如果给出了参数,则以给出的参数为准,例如:

```
#include <iostream.h>
void display (int x=1000)
{
    cout<<x<<endl;
}

void main()
{
    display(500); //函数在调用时给出了参数500,这时x的参数以500为准,程序输出为500
}
```

一个函数可以有多个默认参数,但是,所有的默认参数必须列在参数表的最后。例如,以下的函数原型定义是错误的。

```
void fun(int a , int b=1, int c, int d=2); //此写法是错误的
```

在函数调用时参数 a, c 不能缺省,而 c 在可默认参数 b 的后面,所以产生错误。而下面的函数原型定义是正确的。

```
void fun(int a ,int b ,int c=1,int d=2);
```

因为在函数调用时参数 a, b 是不能缺省的,而 c、d 为默认参数,所有的默认参数必须列在参数表的最后。换句话说,调用函数时,参数应连续给出,不能间断。

例如,若有函数原型定义:

```
void fun(int a=0, int b=0, int c=0, int d=0);
```

那么,下面的各种调用形式都是正确的。

```
fun();
fun(1);
fun(1,2);
fun(1,2,3);
fun(1,2,3,4);
```

1.7 引用参数

引用主要用来向函数传递参数,以及从函数中返回值。引用指对变量或对象取一个别名。也就是说引用和原变量共用一个地址,把某个变量的地址看作是该变量的别名。当初初始化一个引用时就将它和一个变量联系起来,这个引用将一直与变量相联系,以后

就不能将它改变为其他变量的引用。引用可应用在如下方面。

- (1) 独立引用。
- (2) 参数传递。
- (3) 传回引用。

引用的声明：

类型 &引用名=变量名；

例如：

```
int &x
```

可读成 x 是对一个整型变量的引用。例如：

```
int num;
int &x=num;
x++;           //用 num 的别名 x 把 num 自增 1
```

这两条语句声明了一个名为 num 的整数。又声明 x 是对 num 的引用。以后对作用于这两个名字上的所有操作都具有相同的结果。

1.7.1 独立引用

为了加深对独立引用的理解，首先让我们阅读以下程序。

例 1.6 独立引用的例子。

```
#include <iostream.h >
void main()
{
    int num=50;
    int &x=num;
    x=x+10;
    cout<<"num="<<num<<" ";
    cout<<"x="<<x<<endl;
    num=num+20;
    cout<<"num="<<num<<" ";
    cout<<"x="<<x<<endl;
}
```

程序的运行结果为：

```
num=60    x=60
num=80    x=80
```

从上例中可以看出：作用于 num 的操作，实际上就是作用于引用 x，它们是指同一个变量，只不过 x 是 num 的别名而已。语句“int &x=num;”在执行时并没有在内存中建立一个新的变量 x，而只是告知编译器 num 又有了一个名字 x。

引用和指针非常相似，但也有区别。指针变量的内容是一个地址，而引用为变量的别名。