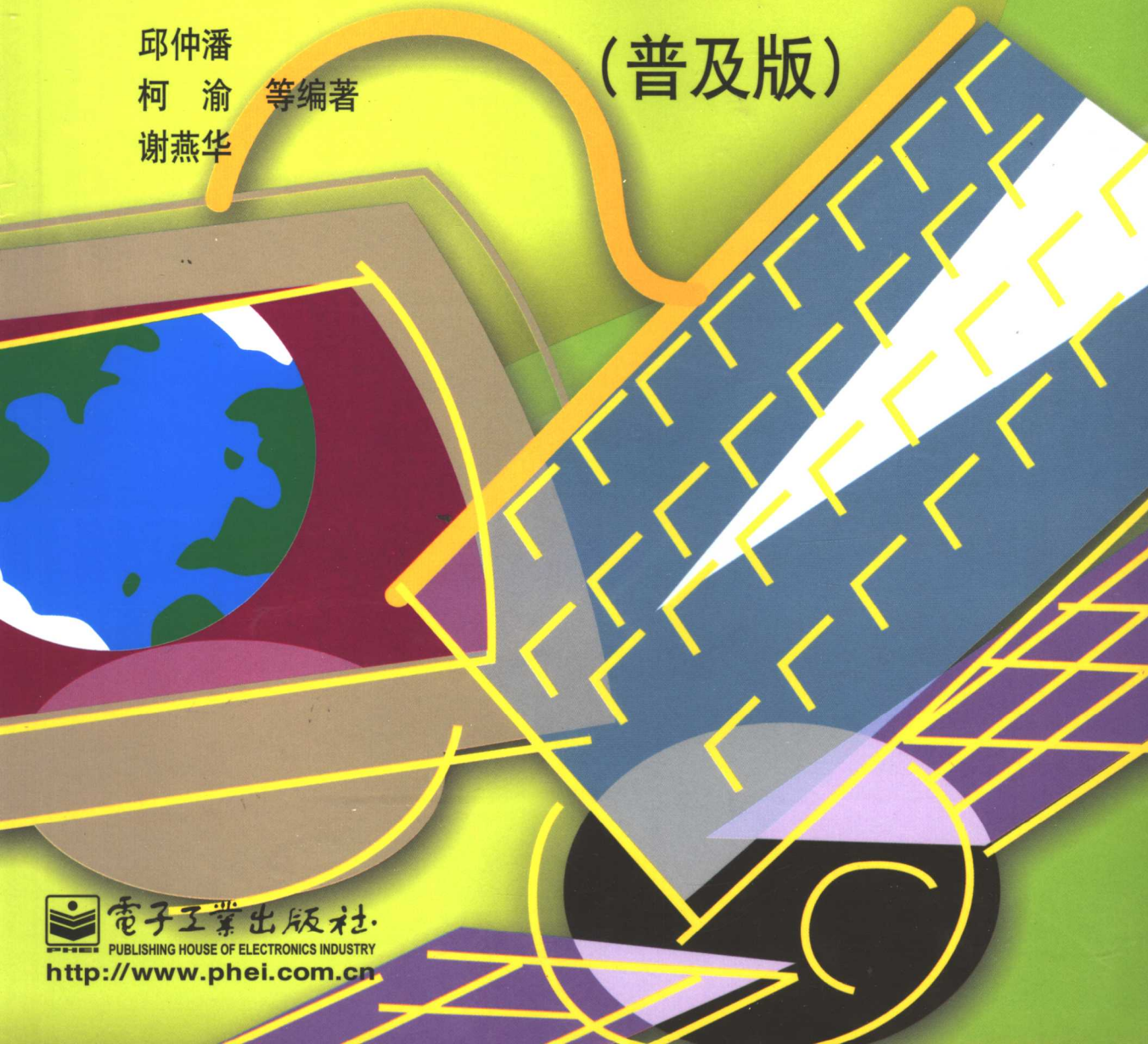
Visual C++ 6

从入门到精通

（普及版）

邱仲潘
柯　渝　等编著
谢燕华



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Visual C++ 6从入门到精通（普及版）

邱仲潘　柯　渝　谢燕华　等编著

電子工業出版社
Publishing House of Electronics Industry
北京・BEIJING

内 容 简 介

本书全面介绍了Visual C++ 6的特点、使用方法及编程技巧，旨在提供Visual C++的“从入门到精通”式的综合性指南。具体内容包括：Visual C++集成式编程环境，Visual Studio、Windows NT/98 GUI编程，微软基础类、应用程序向导、类向类、类库和ActiveX控制的使用，以及文件访问和图形打印等。

本书为“从入门到精通”类图书，适合阅读的读者范围涵盖初学者到高级技术人员。

图书在版编目（CIP）数据

Visual C++ 6从入门到精通（普及版）/邱仲潘等编著.—北京：电子工业出版社，2005.9

ISBN 7-121-01377-0

Ⅰ. V… Ⅱ. 邱… Ⅲ. C语言—程序设计 Ⅳ. TP312

中国版本图书馆CIP数据核字（2005）第096284号

责任编辑：李 莹

印 刷：北京天竺颖华印刷厂

出版发行：电子工业出版社

北京市海淀区万寿路173信箱 邮编：100036

北京市海淀区翠微东里甲2号 邮编：100036

经 销：各地新华书店

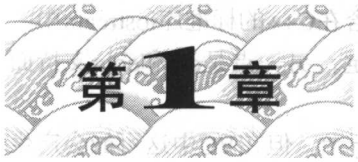
开 本：787×1092 1/16 印张：29.25 字数：730千字

印 次：2005年9月第1次印刷

定 价：40.00元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换，若书店售缺，请与本社发行部联系。联系电话：010-68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

目　录



C++的特性

Visual C++是一个功能非常强大的可视化应用程序开发工具，是计算机界公认的最优秀的应用开发工具之一。Visual C++是一种程序设计语言，同时也是一个集成开发工具，提供了软件代码自动生成和可视化的资源编辑功能。在使用Visual C++开发应用程序的过程中，系统为我们生成了大量的各种类型的文件。在介绍VC++的各种功能之前，先用3章介绍C++基础知识。如果读者已经熟悉这些内容，可以跳过；如果读者需要学习更多C/C++的知识，可以参考有关书籍。

C++现在得到了越来越广泛的应用，它继承了C语言的优点，并具有自己的特点。其中有些特点也出现在新版的C语言中，例如内联函数和常量类型等。本章主要介绍这方面的内容。

1.1 输入与输出

在C++中，除了可以使用C的外部库函数，比如printf和scanf等进行输入输出外，还提供了通过cin和cout的输入输出方式。比如，下面两个程序在C++中都是适用的：

```
// 程序1
#include "stdio.h"

void main()
{
   int a, b, sum;
   scanf("%d", &a);
   scanf("%d", &b);
   sum = a + b;
   printf("sum is %d", sum);
}

// 程序2
#include <iostream>

void main()
{
   int a, b, sum;
   Dcin>>a>>b;
   cout<<"sum is : "<<sum;
}
```

cin和cout的优点是可以自动调整输入和输出格式，它们包含在C++的标准库iostream中。它们不必像printf和scanf那样，格式一个个都需要由使用者指定。另一点是cin和cout支持面向对象的输入和输出。

对C++标准库的头文件的引用不必加上.h，如上面的代码所示。但在VC中这可能会产生编译错误。因此可以采取下面两种写法：

```
// 仍然加上.h
#include <iostream.h>
```

或者

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

1.2 注释语句

在C++程序中，除了可以使用标准C的注释分隔符（/*和*/）外，还可以使用单个（//）字符进行注释，该注释从“//”开始，到该行尾结束。因此，如下两个语句是等价的。

```
x++ ; /* This is a comment */
x++ ; // This is a comment
```

C++的“//”注释方式对于单行注释非常适用，显得很简洁；但当注释是多行的时候，/*和*/的注释方式就显得简洁，如果要使用“//”的注释方式，则每行都要以//开头。注意，/*……*/的注释方式不能嵌套。

1.3 声明语句

在C语言中，变量必须在程序块的开头声明，全局变量必须在任何函数之前声明，局部变量必须在程序块的所有执行语句之前声明。而在C++中，变量的声明非常灵活，只需满足在引用前声明这一条件。C++把变量声明语句放在所使用的代码之前，使代码的阅读和维护变得容易。例如在C语言中，下面这个程序段是错误的：

```
Sample()
{
  int i ;
  i = 10 ;
  Dint j ;  // 在c中是错误的
  j = 45 ;
  // .....
}
```

因为变量j没有在程序块的所有执行语句之前声明，C语言的编译器会提示出错，并停止编译；但是在C++中，以上程序段是正确的。在C++中，甚至可以在for语句中声明变量：

```
// other statements
for (int i = 0; i < 100; ++i)
{
  // other statements
}
```

通常认为，对于大函数，在最靠近使用变量的位置声明变量是合理的；而对于较短的函数，在函数开头部分声明变量是比较好的做法。

1.4 作用域操作符

内部范围的局部变量和外部范围的变量同名时，内部范围的局部变量会使外部变量失去作用。看一下下面的代码：

```
double a ;
void Func ()
{
  int a ;

  a = 5 ;
}
```

在上面的例子中，Func函数中的int a声明使外部的double a声明失去作用，使得函数内的赋值改变了int变量，而不是double变量。

在C++语言中，被同名局部变量掩盖的全局变量也可以访问到，只需要在变量名前面加上作用域操作符，如以下代码所示：

```
double  a ;//global variable a
void Func()
{
  int a ;          //local variable a

  a = 5 ;          //assigns 5 to local int a
  :: a = 4.5 ;     //assigns 4.5 to global double a
}
```

注意，作用域操作符只能用于访问全局变量，不能用它访问操作块中声明的局部变量。下列代码就是错误的：

```
void Func()
{
  double a ;
  int count = 1 ;

  if (count = = 1)
  {
    int a ;
    //other statements
```

```
        :: a = 2.5 ;            //ERROR: :: specifies global variable
        //other statements
    }
    //other statements
    }
```

也可以在作用域操作符访问被同名局部类型或枚举掩盖的全局类型或枚举。

1.5 内联函数

在C中我们常常用到宏，比如声明一个计算平方的带参数的宏：

```
#define square(x) x * x
```

当用户调用宏square(2)时，编译程序就会解释为：

```
2 * 2
```

但是宏在使用时可能会有不可预料的情况发生，比如用户在调用square(x + 1)时，本意是希望计算（x + 1）*（x + 1）的值，但在宏展开时，变成了：

```
x + 1 * x + 1
```

C++中引入的内联函数与C中的宏作用类似，但不会出现上面的问题。用inline关键字来声明内联函数。何谓内联函数呢？在编译过程中，调用内联函数时，编译器使用函数体的代码插入到调用该函数的语句处，这样在程序运行的时候就不需要进行函数调用。与非内联函数不同的是，如果改变了一个内联函数，则调用它的所有的源文件都要重新编译。

为什么引入内联函数这个概念呢？主要是为了消除函数调用时带来的系统开销。函数的内联化并不改变其含义，只是影响所产生代码的速度和长度，因此在调用地方比较少、代码长度比较短的函数中可以用inline关键字定义，特别是循环调用时，可以消除大量系统开销。内联函数在第一次被调用之前必须定义或声明。例如，用内联函数设计求平方函数：

```
inline double square(double x)
{
   return x * x;
}
```

内联函数体一般都比较小。通常情况下，内联函数不包含各种复杂的循环控制语句，函数体的代码在5行以内为宜，否则会使编译后的程序代码增大。

1.6 默认函数参数

一般来说，实参个数和形参个数一样多，但是C++中可以在声明函数时定义默认参数值。当函数调用中省略默认参数时，默认参数值自动传递给被调用函数。

默认参数必须是函数参数列表中最右边的参数，那么该参数右边的所有参数都必须省略掉。默认参数要在第一次出现该函数名的地方指定，通常在函数原型中指定。下面的程序

在计算长方体体积时使用了默认参数的用法。

```
#include <iostream>

using namespace std ;

int boxVolume(int length = 1, int width = 1, int height = 1)
{
        return length * width * height ;
}

  // 非法的默认函数参数声明
  int boxVolume2(int length = 1, int width, int height = 1)
  {
        .......
  }

void main(void)
{
        cout << "The default box volume is :"
                << boxVolume()
                <<'\n'
                <<"The box volume of the box with"
<<"length 10 ,width 1, height 1 is: "
                <<boxVolume(10)
                <<'\n'
                <<"The box volume of the box with"
<<"length 10,width 5, height 1 is :"
                <<boxVolume(10,5)
                <<'\n'
            <<"The box volume of the with length"
<<"10, width 5, height 2 is :"
                <<boxVolume(10,5,2)
            <<'\n'
            <<endl ;
        return ;
}
```

输出结果为：

```
The default box volume is :1
The box volume of the box with length 10 ,width 1, height 1 is: 10
The box volume of the box with length 10,width 5, height 1 is :50
The box volume of the with length 10, width 5, height 2 is :100
```

1.7 引用参数

在C语言中，所有的函数调用都是传值调用，传引用调用是通过模拟实现的，即通过传

递指向对象的指针并在函数调用时通过复引用该指针访问这个对象。

引用参数是其相对应的函数变量的别名。要声明为引用参数，可以用&操作符声明引用。例如：

```
int count = 0 ;
int &refCount = count ;
```

定义引用参数时必须将它初始化，不能用同一个引用参数引用不同的变量，也不能用常数值来初始化引用参数。

下面的程序比较了传值调用、使用指针的传引用调用和使用引用参数的传引用调用。

```
#include<iostream>
#include <stdio.h>

using namespace std ;

int addByValue(int) ;
int addByPointer(int *) ;
int addByRef(int &) ;

void main(void)
{
   int x = 1 ;
   int y = 2 ;
   int z = 3 ;

   cout << "x = " << x << " before addByValue.\n"
        <<"Value returned by addByValue: "
        <<addByValue(x) <<'\n'
        << "x = " << x << " after addByValue.\n"<< endl ;

   cout << "y = "<< y << " before addByPointer.\n" ;

   cout <<"Value returned by addByPointer: "
        <<addByPointer(&y) << '\n' ;

   cout << "y = " << y << " after addByPointer.\n" << endl ;

   cout << "z = " << z << " before addByRef.\n" ;
   cout <<"Value returned by addByRef: "
        <<addByRef(z) << '\n' ;
   cout << "z = " << z << " after addByRef.\n"
        << endl ;

   return ;
}

int addByValue(int a)
{
   a = a + 1 ;
   return a ;
```

```
}
int addByPointer(int *a)
{
   *a = *a + 1 ;
   return *a ;
}
int addByRef(int &a)
{
   a = a + 1 ;
   return a ;
}
```

输出结果为：

```
x = 1 before addByValue.
Value returned by addByValue: 2
x = 1 after addByValue.

y = 2 before addByPointer.
Value returned by addByPointer: 3
y = 3 after addByPointer.

z = 3 before addByRef.
Value returned by addByRef: 4
z = 4 after addByRef
```

1.8 const限定符

const限定符可以用来声明所谓的“常量变量”，例如：

```
const int MAXNUM = 100 ;
```

定义const常量时，一定要进行初始化，不能用赋值改变其数值。可以用常量表达式或变量初始化const常量，以下几种声明都是有效的：

```
int a = 1 ;
const int A1 = 2 ;
const int A2 = a ;
const int A3 = 'a' ;
const int A4 = 2 + 1 ;
```

常量变量可以用于任何要求使用常量表达式的地方。例如，数组声明用常量变量制定数组的大小：

```
const int arraySize = 100 ;
int a[arraySize] ;
```

常见的程序设计错误有：

1. 用非常量表达式初始化常量指针。
2. 试图修改常量变量。
3. 试图修改常量指针。

1.9 函数重载

在C语言中，在同一个程序中声明两个同名的函数是一种语法错误。但是在C++中，只要函数定义了不同的参数集，就可以用同一名字定义这些函数，称为“函数重载”。在调用一个重载函数时，C++的编译器通过检查参数的个数、类型和顺序自动选择一个合适的函数。例如，可以声明两个版本的函数Abs，一个取得int的绝对值，一个取得double的绝对值：

```
#include <iostream>
#include <stdio.h>

using namespace std ;

int Abs(int a)
{
        if (a > 0)
        {
                return a ;
        }
        else
        {
                return -a ;
        }
}

double Abs(double a)
{
        if(a > 0)
        {
                return a ;
        }
        else
        {
                return -a ;
        }
}

void main(void)
{
        int i ;
        double d ;

        i = Abs(5) ;
        d = Abs(-0.25) ;
```

```
    cout << "i = " << i << endl ;
    cout << "d = " << d << endl ;
    return ;
}
```

输出结果为：

```
i = 5
d = 0.25
```

函数重载能够在多种数据类型的基础上执行任务，有助于省去使用C中的#define宏。重载函数能够利用C++的类型检查功能，而在C语言中使用宏时没有这一功能。

要重载的函数必须在参数个数或者参数类型上有所不同，否则编译系统将不知道如何选择函数，也不能以函数返回值的不同作为重载的依据。比如，下面的函数重载是非法的：

```
// 不能以返回值的不同作为重载依据
int Abs(int a);
double Abs(int a);
```

函数重载有时候也会因为类型转化而产生错误，例如下面的例子：

```
int Abs(int a);
long Abs(long a);
.....
// 3.12不知道应该转化为int还是long
Abs(3.12);
```

1.10 new和delete操作符

C++中的new和delete操作符可以使程序实现动态内存分配。语句

```
ptr = new typename;
```

用于从程序的空闲内存区中为typename类型的对象分配内存。new操作符自动建立一个具有合适大小的对象，返回具有正确类型的指针。如果内存分配不成功，返回一个空指针。

在C++中用如下语句释放该对象所占用的内存：

```
delete ptr ;
```

delete操作符只能释放由new操作符分配的内存。在程序的执行过程中，把delete用于已经释放的指针将导致无法预料的错误。

可以用new动态地建立数组。下述语句动态地分配了一个有200个整型元素的一维数组，并把new操作符返回的指针赋给整型指针arrayptr：

```
int *arrayptr ;
arrayptr = new int [200] ;
```

要释放为该数组分配的内存，使用如下语句：

```
delete [] arrayptr ;
```

new和delete操作符通常比传统的malloc系列内存分配函数更有用。在使用new时自动计算要分配类型的大小，而不使用sizeof操作符，也避免了分配错误的存储量。同时，new可以自动返回正确的指针类型，而不必对返回指针进行类型转化。在使用类对象时，new还可以自动调用类构造器（类的构造函数），delete自动调用类销毁器（类的析构函数）。

1.11 C++的模板

模板是C++语言中的一个重要特性。使用C++模板能够方便地生成操作各种不同数据函数的族和类族，从而避免为各种类型写不同的函数和类。本节主要介绍函数模板的编写和类模板的编写。

1.11.1 函数模板

如果要将一个函数用于各种函数类型，则可利用函数重载来实现。回顾一下本章函数重载的例子：

```
int Abs(int a)
{
    if (a > 0)
    {     return a ;      }
    else
    {     return -a ;     }
}

double Abs(double a)
{
    if(a > 0)
    {  return a ; }
    else
    {     return -a ;     }
}
```

但是如果使用C++的函数模板，则可以只写一个定义，自动处理各种类型的数据。函数模板的一般说明形式如下：

```
template <class T>
T func (T N)
{
.....
}
```

其中，template是声明模板的关键字。 <class T>中的T表示类型参数，这里的class不是代表类，而是表示任何类型的意思；T func(T N)中的第一个T表示返回类型，func表示函数名，T N表示模板形参表。据此，上面的重载函数可以转化成函数模板：

```
template <class T>
```