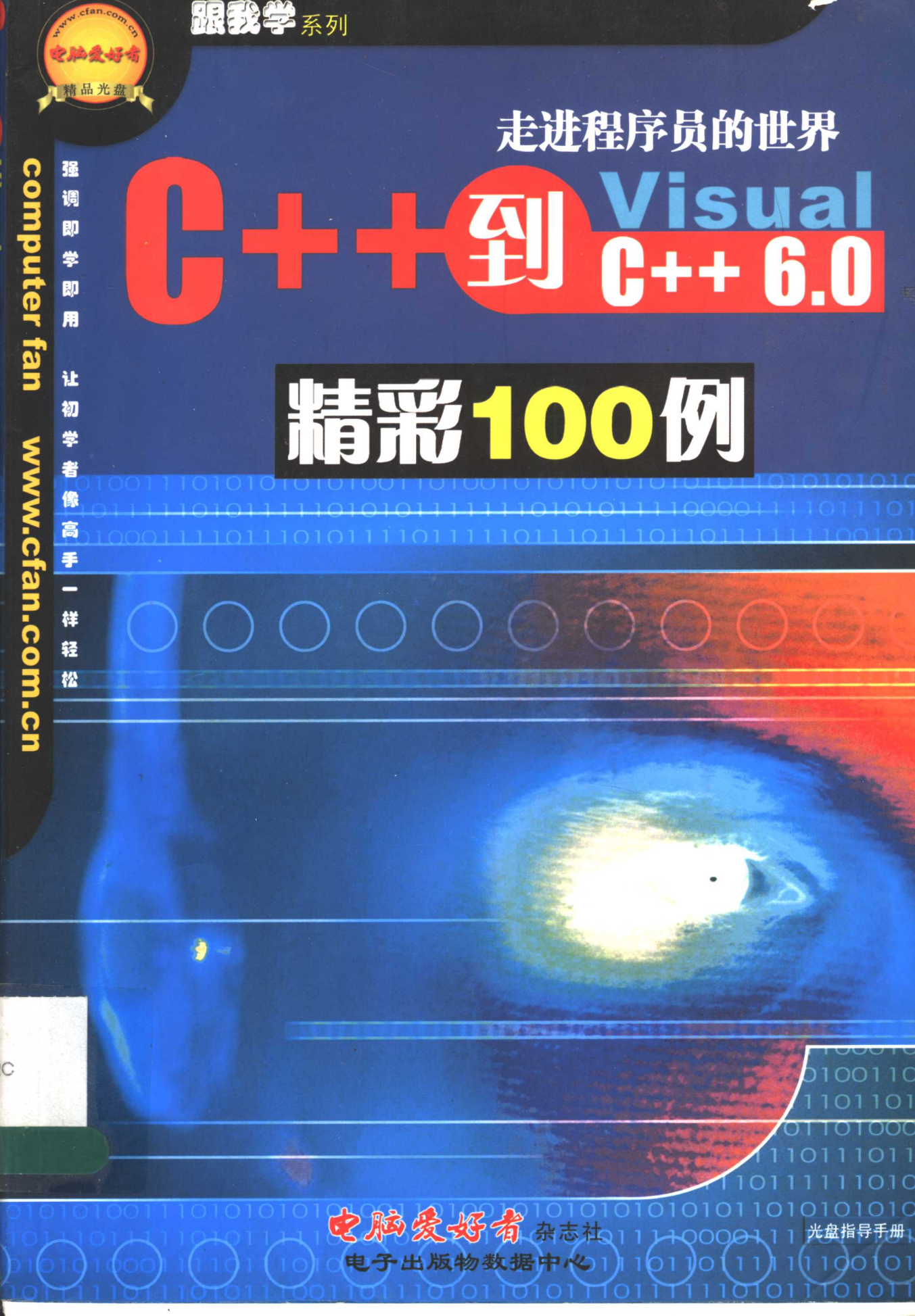
跟我学系列

走进程序员的世界

C++到Visual C++ 6.0

精彩100例

强调即学即用 让初学者像高手一样轻松



电脑爱好者杂志社
电子出版物数据中心

光盘指导手册

C++到Visual C++ 6.0精彩100例

作　　者：应甫臣　刘迎辉
责任编辑：顾建林

《电脑爱好者》杂志社

2001年·北京

内 容 介 绍

本书通过 100 个实例介绍 C++程序设计语言和 VC++开发环境的使用技巧。内容划分成二大部分八个章节：第一部分是 C++程序设计经验谈，第一章是 C++语言的简介、与 C 语言相比较，它们的相似之处及其继承的优点。第二章是关于如何编写出优质的 C++程序而写的二十条经验之谈。第二部分是 Windows 程序设计的 100 个实例，内容分为：掌握程序语言的特点、Windows 系统的基本技术、用户界面编程、多媒体编程、数据库与 Internet、算法共六章。涉及了 C++语言特性的具体使用，阐述了使用 C++进行程序设计的基本思路和方法。使用 C++和 VC++进行 Windows 应用程序设计的各个方面，具体包括：界面设计，系统监控，多媒体应用，数据库连接，网络开发和算法研究。在各个章节中，安排一到二个比较复杂的实例，阐明了软件设计的简单流程。

系列光盘配套图书：跟我学系列
书　　名：《C++到VC++精彩100例》
作　　者：应甫臣　　刘迎辉
策　　划：顾建林
责任编辑：顾建林　　审　　校：廖宇雷

编辑出版：北京《电脑爱好者》杂志社
发行单位：北京《电脑爱好者》杂志社
地　　址：北京9615信箱(100086)
电　　话：010-62144153（直拨）、62161335、62161337转8417
网　　址：http://www.cfan.net.cn
技术支持：010-62143106

开　　本：787×1092　1/16　　印张：17
版　　次：2002年1月第1版　　2002 年 1 月第 2 次印刷
字　　数：500千字
印　　数：10 001—20 000 册
本 版 号：ISBN 7-89999-636-8/TP・254
定　　价：28.00元（1CD，含配套手册）

前　言

自从程序设计进入面向对象时代，领导这一潮流的 C++程序设计语言就不可避免的成为广大程序员和程序设计爱好者的第一选择。微软的可视化开发环境 Visual C++凭借其良好的界面、强大的功能和对 Windows 的最好支持获得了广泛的认同。市场上介绍 C++和 Visual C++的书籍比比皆是，其中不乏精品，但是，读者不难发现大部分书籍的侧重点有的是说教式的语法讲解，有的是帮助文件式的功能分析，还有的就是讲解开发环境的使用指南。

读者现在看到的这本书与上述书籍有所不同，书籍不是讲解 C++的基本概念和编程方法，也不是机械的介绍 Visual C++的基本使用方法和功能。本书将通过精心选取的 100 个深入浅出的实例，为读者介绍一些程序设计和软件开发过程中经常使用的技巧，同时也将阐述一些具有实际应用价值的开发方法。通过对本书的阅读，读者不仅可以更加深入的理解 C++程序设计语言以及 Visual C++开发环境的特性，而且可以更加灵活的处理软件开发过程当中，所遇到的各种问题。

书的内容划分成二大部分八个章节：第一部分是 C++程序设计经验谈，第一章是 C++语言的简介、与 C 语言相比较，它们的相似之处及其继承的优点。第二章是关于如何编写出优质的 C++程序而写的二十条经验之谈。作为第一部分的延续，第二部分是 Windows 程序设计的 100 个实例，内容分为：掌握程序语言的特点、Windows 系统的基本技术、用户界面编程、多媒体编程、数据库与 Internet、算法共六章。涉及了 C++语言特性的具体使用，阐述了使用 C++进行程序设计的基本思路和方法。使用 C++和 VC++进行 Windows 应用程序设计的各个方面。读者除了能够看到各种各样的 Windows 应用程序，还能通过作者详细深入、浅显易懂的介绍、分析学到非常实用的技巧。在各个章节中，安排一到二个比较复杂的实例，阐明了软件设计的简单流程。

与一般书籍不同的是，本书着重分析语言的深层次的特性，讲解程序开发过程中容易混淆和难于理解的知识点，使读者能够更好的将理论转化为实际的编程方法。Windows 实例部分读者可以了解如何使用 C++和 Visual C++开发 Windows 应用程序，介绍了开发技巧的综合应用。读者能够深刻理解掌握软件开发的整体设计思想和开发过程。

书中虽然提供了 100 个实例，但除了几个非常典型的实例，其它的没有在书中放上大量的代码，而只是将关键部分列出来，方便读者对照实例的叙述文字，完整的代码都放在本书的配套光盘中。该书配套光盘所提供的实例源代码都在 Visual C++6.0 环境下调试过，读者可以直接使用，很多代码都能够直接用到读者自己的程序中去。当然，如果你能够详细的阅读分析源代码，并将它更好的加以改善，相信肯定会有更大的收获。

本书在编著的过程中，感谢在工作上支持我的领导、同事以及朋友，特别感谢 nesgood com 网站廖宇雷先生、北京的达利先生的大力支持，对于他们的协助，表示由衷的感谢。

由于作者能力和视野有限，实例的选取，代码的分析，技巧的讲解都可能存在不妥当之处，敬请广大读者朋友批评指正。

责编：顾建林

2001 年 12 月 1 日 于北京

内 容 导 读

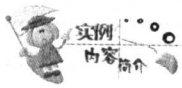
第一部分通过二十个主题讨论了编写优质 C++程序的技巧和经验。虽然这部分内容粗看之下非常简单，但只有当读者真正将这些技巧应用到自己的程序中后，也许才能发现这些内容的真正价值。所以读者即使对本书的其他内容不感兴趣，第一章的内容也是无论如何不应该错过的。

第二部分的 100 个实例，作者分为：掌握程序语言的特点、Windows 系统基本技术、用户界面编程、多媒体编程、数据库与 Internet、算法六个类别。这些类别的实例都是彼此独立的，因此读者也能够随意的选择阅读顺序。光盘上源代码与书中的章节设置也是完全一一对应的，读者可以轻松的找到所需要的内容。

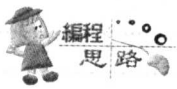
每个实例一般分为“实例内容简介”、“编程思路”和“编程步骤”三部分，并以图标分隔各个部分的文字，以做到一目了然。

整本书的编排格式正文部分中文都是采用五号宋体字、英文采用 Courier New 字体，而代码部分则将字体缩小为小五号字体。

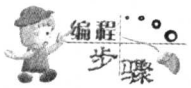
正文和代码的编排格式示例：



在程序中共享数据有许多方法，设置全局性的变量是一种常用的方法。但是，全局变量是有其局限性的。全局变量的全局性使得……



从本节下面实例的输出结果可以看到对于对象 M 和对象 N，Sum 的值……




```
// 这是代码
   class Myclass {
   public:
     Myclass(int a, int b, int c);
   ......
```

目 录

第一篇 C++程序设计经验谈

第一章 C++简介

许多人都认为C++难学！

确实如此，C++之所以难学，不仅在于其灵活的语法、强大的能力、复杂的对象模型等。而且C++提供了多达四种程序设计方式：基于过程、基于对象、对象导向和泛型设计。而在Windows世界中，原本难学的C++加上庞大而繁杂的Windows API更是一种巨大的挑战。但是C++程序员们并没有被吓退，他们非常明白，一旦成功的掌握这些技能以后，自己就会拥有强大的内动力。

虽然在某些领域，C++并没有任何优势可言，但不可否认的是：一些功能强大的应用软件、性能超群的服务器软件、引人入胜的游戏等大部分都是使用C++开发的，而不是别的编程语言。

因此，笔者和很多C++爱好者一样，认为C++仍然是Windows下面威力最为强大的编程语言。我们真诚的希望你在阅读本书所提供的各种实例内容以后，能够跨越C++的重重险阻，迅速提升自己的程序设计能力。做一个成功而快乐的C++程序员。

C++程序设计语言简介

C++是一种面向对象的程序设计语言，它对风靡一时的C语言进行了扩充和改进，以便更好的支持面向对象的程序设计。

C语言的编程风格简洁紧凑，程序书写形式自由、方便灵活；提供丰富的运算符与数据类型；程序员可以直接访问内存，使得操作系统级的程序开发成为可能；另外由于语言的硬件无关性使得程序的跨平台移植成为可能。

但是C语言在流行的同时也逐渐暴露了它的局限性：类型检查机制相对薄弱，程序发生错误不能在编译期间及时发现；程序本身几乎没有对代码重用的支持，原有的代码不能很好的得到利用，导致了程序开发的资源浪费；结构化的设计风格使得程序管理十分困难，程序

复杂性的增加远远超出程序规模的增加。

因为C++是在C语言的基础上发展起来的，所以其语言保留了很多非面向对象的语言特性，及支持原有编程风格和编程模式；与此同时添加对封装性(Encapsulation)，层次性(Hierarchy)，多态性(Polymorphism)的支持，而这三个特性恰恰是面向对象程序设计的根本特性，也是这些面向对象的特性解决了C语言原有的局限性。

在C++中，采用下面的方法支持面向对象的三个特征：

封装：

封装就是将代码与它要操纵的数据放在一起，从而避免外部的程序对数据的影响。在C++中，程序员可以创建一个类（Class）来完成对数据和操作的封装。一个类就是一个对象，它除了具有自己的属性（数据成员）外，还提供了供外部程序调用的方法（成员函数）。可以说C++中的类就是C++提供封装的一种机制。

继承：

继承就是一种方法，通过这种方法，一个对象能够获得另一个对象的特性和能力。具体到C++中，程序员可以从一个类派生另一个类就能够创造具有层次性的对象体系。

多态：

单一接口、多种方法是多态的形象描述。在C++中，通过重载等技术，程序员能够创建具有相同名字，不同实现方法的函数等。

C++虽然可以进行面向对象编程，但并不是一种纯粹的面向对象语言。这一点被许多人所诟病，但是这个特点恰恰让C++具有了无可比拟的灵活性。C++之所以发展到现在这个样子，在于它有自己的设计目标。理解了这些设计目标，就不难弄懂所有的这些问题了。

C++最首要的目标在于：

和C的兼容性。很多很多的C还存在，很多很多的C程序员还存在。C++利用了这一基础，并建立在这一基础之上。

效率。C++的设计者从一开始就清楚地知道，要想把C程序员争取过来，就要避免转换语言将带来性能上的损失，否则C语言的拥护者们不会对C++再看第二眼。结果证明C++在效率上的确可以和C匹敌 —— 二者相差大约在5%之内。

与传统开发工具及环境的兼容性。不同的开发环境到处都是，编译器、链接器和编辑器也无处不在。从小型到大型的开发环境，C++都需要轻松应对，所以携带的包袱越轻越好。想移植C++吗？你实际上移植的只是一种语言，并利用了目标平台上现有的工具。

解决实际问题的可应用性。C++没有被设计为一种完美的，纯粹的语言，不适用于教学生如何编程。它是为专业程序人员而设计的强大开发工具，用来解决各种领域中的实际问题。

第二章　编写优质的 C++程序

一个程序员能够记住所有的关键词、API、库函数并不代表他就能够写出优秀的 C++程序。要想写出优秀的 C++程序，必须经过长期艰苦的学习磨练方能达到。

本书不是一本告诉大家怎样编写优质 C++程序的教学书，作者自认也没有这样的能力与资格。作者只是作为一个普通的 C++爱好者，为读者收集了许多能够提高 C++程序质量的资料，精心整理后再呈现到读者面前。

就像学英语首先要学好语法一样，编写优秀的 C++程序也要从最基本的地方做起。

初看这个第二章的开场独白看起来是那么短小，内容是那么简单基础，但是请不要轻视它，它将从根本上影响你的 C++程序质量、方法与技巧。即使你自认为是 C++高手，也应该仔细阅读这一节，了解自己是不是因为站得太高而不了解地基是否坚固。

2.1 充分利用 C++的优点

许多 C++程序员都是先学习的 C，然后再是 C++。这样虽然使得学习曲线不那么陡峭，却为以后编写 C++程序造成了许多问题。

```
#include <stdio.h>
void main()
{
  printf("Hello, World !\n");
}
```

上面的代码虽然能够很好的运行，但是它不是一个真正的 C++程序。真正的 C++程序应该具有 C++的特点，而使用 I/O stream 代替 stdio 就是最基本的一点。

在 C++程序中，I/O stream 被设计来进行基本的 I/O 操作。它不但能够完成 stdio.h 能够完成的一切任务，而且对于 C++而言还具有更多的意义。

因为通过重载 I/O 算子，C++程序就能够用统一的方式进行各种类型数据的输入输出。而使用 stdio 时，程序员自定义的类型是无法用标准的方式进行输入输出的。而且更为重要

的是 I/O stream 是类型安全的，对于没有准备相应 I/O 算子的自定义类型，编译器会立刻发现。

虽然上面的程序中使用 stdio.h 没有什么不妥的地方，但是随着程序的增大，你会发现 stdio 的局限性越来越大，以致你最后不得不修改每一个使用 stdio 的地方。

与其后来再修改，还不如一开始就采用 C++的方式来解决问题。

修改后的代码如下：

```
#include <iostream.h>
void main()
{
  cout << "Hello, World !\n";
}
```

读者也许会怀疑在 Windows 中编程还有多大的必要使用控制台输入输出，不是一切都是图形化的吗？

当然，在今天的 Windows 世界中，绝大多数应用程序都通过窗口来与用户进行交互、表示数据。但笔者讲这个例子的目的是要让使者明白，既然你用的是 C++，就要充分利用 C++的优点，仅此而已。

2.2 使用 C++风格的注释

C++包含两种不同的注释风格，即原有 C 的注释风格和新增的 C++注释风格。

C 风格的注释使用一对匹配的斜线和星号，斜线在外面，星号在里边，例如：

```
/* this is a comment! */
```

而下面的注释与上面的等价：

```
/*
this is a comment!
*/
```

可见这种风格的注释将/* 和 */之间的所有内容都看作是注释内容。所以这种注释可以跨过多行，极大的方便了我们在调试程序的时候将大段的代码暂时从程序中屏蔽掉，以便发现程序的问题所在。

C++风格的注释使用双斜线作为注释开始的标志，例如：

```
// this is a comment!
```

这种注释风格不需要结束标志，行结束符自动终止本行注释。

C 注释语法在 C++里仍然可以用，C++新发明的行尾注释语法也有其过人之处。例如下面这种情形：

```
if ( a > b ) {
  int temp = a; // swap a and b
  a = b;
  b = temp;
}
```

从软件工程的角度看，写这段代码的程序员也做得很好，他最初的代码里也写了一个注释，以解释代码在做什么。假设你出于某种原因要注释掉这个代码块，使用 C 风格的注释就会让你遇到麻烦：

```
if ( a > b ) {
  /*  int temp = a;  /* swap a and b */
    a = b;
    b = temp;
  */
}
```

请注意嵌套在代码块里的注释是怎么无意间使本来想注释掉整个代码块的注释提前结束的，而改为 C++风格的注释就不会产生这样的问题了。

```
if ( a > b ) {
  /*  int temp = a;  / / swap a and b
    a = b;
    b = temp;
  */
}
```

C 风格的注释当然还有它存在的价值。例如，它们在 C 和 C++编译器都要处理的头文件中是无法替代的，而且可以看出新的风格对于短小的，仅有一行的注释比较方便，但在处理比较长的注释时就不太方便了。

但是我们仍然要记住，只要有可能，尽量使用 C++风格的注释。

2.3 使用 const 关键字

使用 const 关键字表明某种对象不能被修改，内部的实现将由编译器来实施。通过

const 关键字，你可以通知编译器和其他程序员某个值要保持不变。

const 关键字具有广泛的作用空间，在类的外面，它可以用于全局对象和静态对象；在类的内部，它可以用于静态和非静态成员。

对指针来说，const 关键字带来的变化尤其显得神奇有效，可以指定指针本身为 const，也可以指定指针所指的数据为 const，或二者同时指定为 const，当然两者都不指定为 const 也是经常用到：

```
char *p = "hello";                    // 非 const 指针，非 const 数据
const char *p = "hello";              // 非 const 指针，const 数据
char * const p = "hello";             // const 指针，非 const 数据
const char * const p = "hello";       // const 指针，const 数据
```

我们可以通过星号(*)和 const 关键字的位置关系来判别语句的含义，如果 const 出现在“*”的左边，指针指向的数据为常量；如果 const 出现在“*”的右边，指针本身为常量；如果 const 在两边都出现，二者都是常量。

const 关键字在函数声明中同样具有重要的意义。在一个函数声明中，const 可以指的是函数的返回值，或某个参数；对于成员函数，还可以指的是整个函数。

让函数返回一个常量值经常可以在不降低安全性和效率的情况下减少用户出错的概率。返回值使用 const 有可能提高一个函数的安全性和效率，否则还会出现问题。

2.4 使用 new 和 delete 而不是 malloc 和 free

使用 malloc 和 free 没有什么不好，除非你不是在编写 C++程序。

在 C++世界中，你必须时刻知道对象的存在。声明一个对象时，虽然你没有明确要求，但是这个对象的构造函数却被调用了，而销毁一个对象时，对象的析构函数将被调用。

而遗憾的是 malloc 和 free 是 C 里面的东西，它们对 C++中的一切一无所知。因此当你用 malloc 为一个对象分配内存时，由于不会调用对象的构造函数，使得这个对象处于没有初始化的状态。也许有些对象确实不需要初始化，但绝大部分对象是需要的。

也许你能够记得什么时候用 malloc，什么时候用 new，不过笔者认为与其在 malloc 和 new 之间不断纠缠，还不如用 new 和 delete 来代替 malloc 和 free 为好。

当然，C++在继承 C 的特点时也继承了 C 庞大的函数库，这些库中的内存分配大部分是使用 malloc 和 free 完成的。因此当你在 C++程序中使用这些库时必须记得以 free 对付

malloc 分配的内存，而不是用 delete。

2.5 尽可能推迟变量的声明

```
int strlen(const char* str)
{
  int nLen;
  if(str == NULL) return 0;
  nLen = 0;
  while(str[nLen] != 0) nLen++;
  return nLen;
}
```

这个函数工作起来很正常，但是却有一点小小的遗憾。

如果传递给 strlen 的参数是一个 NULL，则 while 循环根本就不会执行，而我们提前声明的 nLen 就白白浪费了。

所以我们可以将上面的代码修改一下：

```
int strlen(const char* str)
{
  if(str == NULL) return 0;
  int nLen;
  nLen = 0;
  while(str[nLen]) nLen++;
  return nLen;
}
```

现在看上去要好多了，如果 str 等于 NULL，则程序不会浪费时间去为 nLen 分配内存。

不过这个函数仍然有改进的余地：

```
int strlen(const char* str)
{
  if(str == NULL) return 0;
  int nLen = 0;
  while(str[nLen]) nLen++;
  return nLen;
}
```

这次我们在声明 nLen 时就让其拥有初值。

虽然上面这个简单函数并没有从我们的修改中得到什么好处，但读者只要考虑一下如果nLen不是一个int，而是一个像CString这样的复杂类会怎么样。

在使用强度很高的代码中，这些小小的修改有时候能够带来意想不到的效果。

2.6 关于内存申请

大多数情况下，执行动态内存分配的类都在构造函数里用new分配内存，然后在析构函数里用delete释放内存。最初写这个类的时候当然不难做，你会记得最后对在所有构造函数里分配了内存的所有成员使用delete。

然而，这个类经过维护、升级后，情况就会变得困难了，因为对类的代码进行修改的程序员不一定就是最早写这个类的人。而增加一个指针成员意味着几乎都要进行下面的工作：

在每个构造函数里对指针进行初始化。对于一些构造函数，如果没有内存要分配给指针的话，指针要被初始化为NULL(即空指针)。

删除现有的内存，通过赋值操作符分配给指针新的内存。

在析构函数里删除指针。

如果在构造函数里忘记了初始化某个指针，或者在赋值操作的过程中忘了处理它，问题会出现得很快，很明显，所以在实践中这两个问题不会那么折磨你。但是，如果在析构函数里没有删除指针，它不会表现出很明显的外部症状。相反，它可能只是表现为一点微小的内存泄漏，并且不断增长，最后吞噬了你的地址空间，导致程序夭折。

因为这种情况经常不那么引人注意，所以每增加一个指针成员到类里时一定要记清楚在析构函数里面delete掉它。

另外，删除空指针是安全的(因为它什么也没做)。所以，在写构造函数，赋值操作符，或其他成员函数时，类的每个指针成员要么指向有效的内存，要么就指向空，那在你的析构函数里你就可以只用简单地delete掉他们，而不用担心他们是不是被new过。

当然有时候内存的释放工作不是由我们来完成的，就像用智能指针对象时不用劳烦你去删除一样，你也永远不会去删除一个传递给你的指针。换句话说，除非类成员最初用了new，否则是不需要在析构函数里用delete的。说到智能指针，这里介绍一种避免必须删除指针成员的方法，即把这些成员用智能指针对象来代替，比如c++标准库里的auto_ptr。