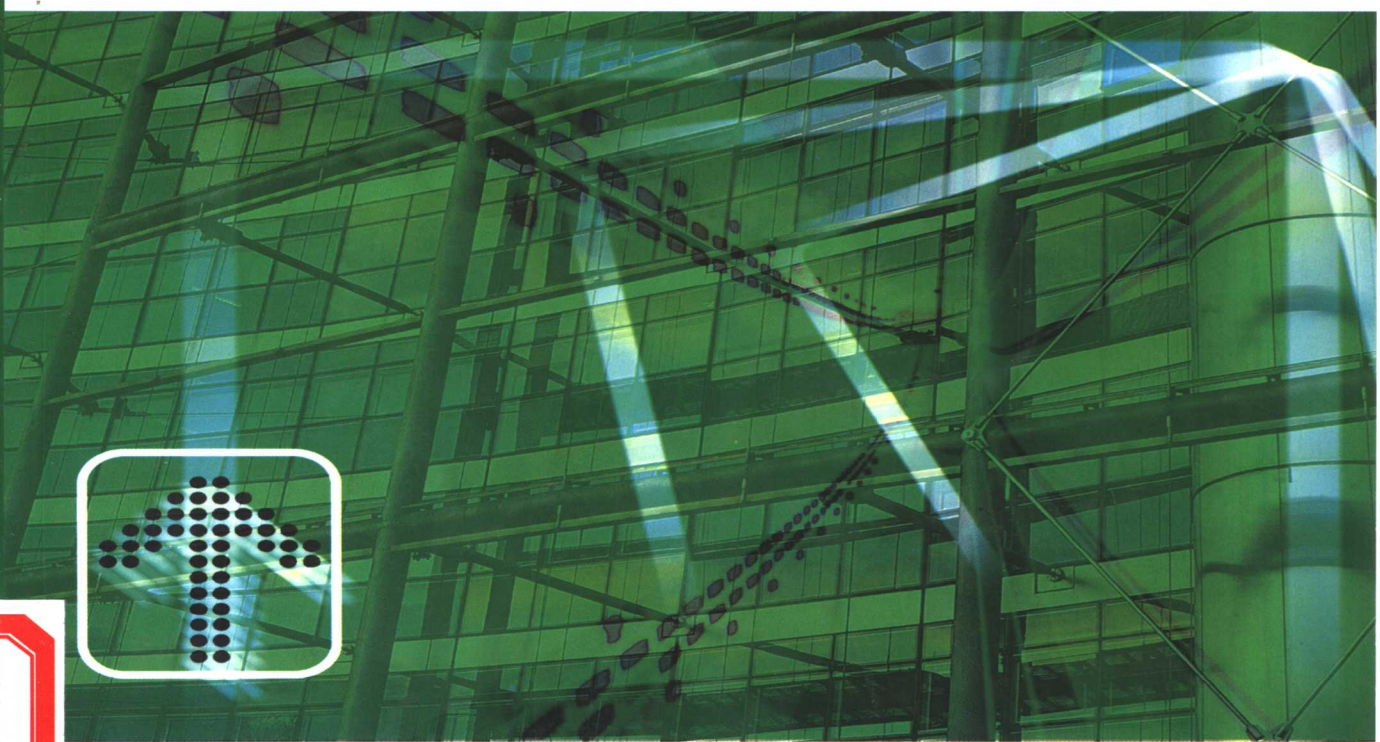
万水软件项目应用与实例开发丛书

COMPANION WEB SITE!

Visual C++ 高级编程及其项目应用开发

郭晓鹏　李存斌　编著



中国水利水电出版社
www.waterpub.com.cn

万水软件项目应用与实例开发丛书

Visual C++高级编程及其项目应用开发

郭晓鹏　李存斌　编著

中国水利水电出版社

内 容 提 要

本书是作者在总结多年 Visual C++开发经验的基础上编著而成的。全书共分两篇。第一部分基础篇共 7 章，结合具体示例详细讲述了 C++基本语法中需要注意的难点问题、Windows 消息及界面设计、进程与线程（包括了一个使用命名管道来实现进程间通信的例子等）、文件操作（包括了内存映射文件、串口读写、INI 文件及注册表文件的操作等内容）、动态链接库以及 COM 编程、Socket 编程以及利用 Visual C++访问数据库等知识点。第二部分项目篇共 5 章，结合一个具体的数据库应用程序——物资管理信息系统，完整地讲述了项目的分析与设计、COM 服务器端应用程序设计、客户端界面设计、物资管理各个业务流程的设计与开发、通用报表及打印模块的设计等部分。在项目部分，通过一个完整的例子，演示了如何利用 COM/DCOM/COM+技术来实现一个三层结构的数据库应用程序、如何利用 ATL 设计对话框应用程序，同时也演示了利用 Visual C++操作 Word 和 Excel 等 Office 应用程序的详细开发过程。读者在具有初步 Visual C++知识的基础上，通过本书的学习，可快速提高 Visual C++的编程能力和实际应用水平。

本书适用于具有初级编程能力的读者，可作为高校高年级学生、研究生的自学参考书和毕业设计的指导书，也可作为相关软件开发人员的参考书。中国水利水电出版社的网站（www.waterpub.com.cn）包括了书中示例和物资管理信息系统的源代码文件，为读者的学习提供了方便，同时也为相关软件开发人员的实际应用开发提供了捷径和参考。

图书在版编目（CIP）数据

Visual C++高级编程及其项目应用开发 / 郭晓鹏等编著. —北京：中国水利水电出版社，2003

（万水软件项目应用与实例开发丛书）

ISBN 7-5084-1730-5

Ⅰ. V… Ⅱ. 郭… Ⅲ. C 语言—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2003）第 087735 号

书　　名	Visual C++高级编程及其项目应用开发
作　　者	郭晓鹏　李存斌　编著
出版、发行	中国水利水电出版社（北京市三里河路 6 号　100044） 网址：www.waterpub.com.cn E-mail：mchannel@public3.bta.net.cn（万水） sale@waterpub.com.cn 电话：（010）63202266（总机）、68331835（营销中心）、82562819（万水）
经　　售	全国各地新华书店和相关出版物销售网点
排　　版	北京万水电子信息有限公司
印　　刷	北京市天竺颖华印刷厂
规　　格	787×1000 毫米　16 开本　29.25 印张　644 千字
版　　次	2004 年 1 月第一版　2004 年 1 月北京第一次印刷
印　　数	0001—5000 册
定　　价	45.00 元

凡购买我社图书，如有缺页、倒页、脱页的，本社营销中心负责调换

丛书前言

随着经济全球化、管理现代化的到来，信息化建设已提到议事日程。国家机关、高等院校、参与国际竞争的大中型企业等单位，为了提高现代化管理水平，为了在激烈的市场竞争中获胜，纷纷不同程度地加强了信息化建设，如组建自己的局域网、实施适用于本单位或本部门的企业级 MIS（管理信息系统）项目或 ERP（企业资源计划）项目。在网络环境下对企业 MIS 或 ERP 项目的开发，对开发语言和工具提出了更高的要求。目前，介绍开发语言和工具的书籍层出不穷，但结合实际应用软件开发项目的书籍并不多。

本套丛书的特点是：结合项目，注重适用性、可操作性和先进性。每本书都按照一个软件项目的开发顺序安排内容，以一个完整的实例（如学生管理信息系统）为主线，将知识点融于实例中，实例的编写结合了作者多年的开发经验，如在录入窗体的按钮设计、查询模块的条件表达式输入等方面都有一定的技巧和经验。

书中源程序均已调试通过，对源程序中难以理解的语句增加了注释说明。源程序除以“程序清单 2-1”方式在书中列出外，相应源程序的一组文件（如项目文件、窗体文件等）均放在网站上，以方便读者学习调用。

本套丛书共有 7 本，内容涵盖了当前流行的可视化软件开发语言和工具，它们是：

- Delphi 深度编程及其项目应用开发
- Visual FoxPro 高级编程及其项目应用开发
- Visual Basic 高级编程及其项目应用开发
- Visual C++高级编程及其项目应用开发
- PowerBuilder 高级编程及其项目应用开发
- ASP 高级编程及其项目应用开发
- Java 高级编程及其项目应用开发

本套丛书适用于具有初级编程能力而急需掌握应用程序设计方法、提高软件项目应用开发水平的读者，而且对于从事软件项目开发的工程技术人员具有一定的参考价值；也可供高校高年级学生毕业设计、研究生课题研究以及教师教学时参考。

李存斌

2003 年 8 月　于北京

前　言

Visual C++是 Windows 环境下最优秀的 C++编译器之一，凭借其友好的集成开发环境、对面向对象技术的全面支持、对 C++标准类库的良好支持、良好的网络及数据库支持、高效的多媒体和图形操作及其高度的灵活性，深受广大软件开发人员的好评。

目前，介绍 Visual C++编程的书籍很多，然而结合一个完整的项目应用程序开发来进行详细讲述的却不多。本书紧密结合项目应用特点，基础篇通过大量示例说明高级编程技术，应用篇以物资管理信息系统的开发贯穿始终。本书是针对不少用户，在具有初步 Visual C++知识的基础上，该如何学习、如何提高自己的编程能力、如何有效掌握 Visual C++开发 Windows 应用程序的技术、如何设计、规划并开发出高质量的应用软件而编写的。通过本书的系统学习，读者可快速提高 Visual C++的编程能力和实际应用水平。

本书是作者在总结多年 Visual C++开发经验的基础上编著而成的。全书共分两篇。第一部分基础篇共 7 章，结合具体示例详细讲述了 C++基本语法中需要注意的难点问题、Windows 消息及界面设计、进程与线程（包括了一个使用命名管道来实现进程间通信的例子等）、文件操作（包括了内存映射文件、串口读写、INI 文件及注册表文件的操作等内容）、动态链接库以及 COM 编程、Socket 编程以及利用 Visual C++访问数据库等知识点。第二部分项目篇共 5 章，结合一个具体的数据库应用程序——物资管理信息系统，完整地讲述了项目的分析与设计、COM 服务器端应用程序设计、客户端界面设计、物资管理各个业务流程的设计与开发、通用报表及打印模块的设计等部分。在项目部分，通过一个完整的例子，演示了如何利用 COM/DCOM/COM+技术来实现一个三层结构的数据库应用程序、如何利用 ATL 设计对话框应用程序，同时也演示了利用 Visual C++操作 Word 和 Excel 等 Office 应用程序的详细开发过程。

本书适用于具有初级编程能力的读者，可作为高校高年级学生、研究生的自学参考书和毕业设计的指导书，也可作为相关软件开发人员的参考书。中国水利水电出版社的网站（www.waterpub.com.cn）包括了书中实例和物资管理信息系统的源代码文件，为读者的学习提供了方便，同时也为相关软件开发人员的实际应用开发提供了捷径和参考。

本书由郭晓鹏、李存斌编著，参与本书编写工作的还有王钇、樊建平、董威、谭显东、李众、朱凯、陈鹏、高丽林等。在本书的编写出版过程中，得到了中国水利水电出版社计算机编辑室全体同志的大力支持和帮助，在此一并表示衷心的感谢。

由于时间仓促和作者水平有限，书中错误和不妥之处在所难免，敬请读者批评指正。

郭晓鹏　李存斌

2003 年 8 月　于北京

目　　录

项目篇——物资管理信息系统项目应用开发

基础篇——Visual C++编程技术

本书的第一部分主要讲述了C++语法中的一些关键问题及其在Visual C++编程中的应用，另外也对MFC的内部机制、多线程、COM编程、网络编程技术、数据库编程技术等进行专门的讲述。

第1章 C++关键语法

C++语言是Visual C++的编程基础。只有熟练掌握C++语言的一些关键语法部分及其重要特性，才有可能更深入地理解和掌握Visual C++程序设计，才可能真正深入地理解MFC的内部机制。

本章将要讲述的C++关键语法部分主要包括关于字符串，指针，类的封装、继承，虚函数与多态，运行时类型识别、类型信息，动态创建，C++中的模板以及异常处理等几个方面。

1.1 C++中的字符串

C++中提供两种字符串表示方法：C风格的字符串和标准C++引入的string类型。在本节中，将对这两种风格的字符串分别进行讲述。

1.1.1 C风格的字符串

C风格的字符串起源于C语言。字符串被存储在一个字符数组中，所有对字符串的操作主要是通过一个char*指针来进行。C++提供了一系列的函数（这些函数包含于标准C库，而标准C库已经成为标准C++的一部分）用于操作字符串。为了使用这些函数，必须包含相应的C头文件：

```
#include <cstring>
```

指向C风格字符串的指针总是指向一个相关的字符数组。即使声明字符串为：

```
char * str = "just a try!";
```

系统也会将该字符串存储到一个数组中去。要想操纵一个C风格的字符串，一般需要用到指针的运算，这为编程带来了一定的难度。在标准C的头文件<cstring>中，为操作字符串提供了不少函数，用以简化对字符串的编程。然而，尽管有不少的字符串函数用以简化编程的

难度，C 风格字符串的底层特性仍然使得对字符串的编程显得非常困难。

先看一段简单的字符串操作代码，如下所示：

程序清单 1-1：

```
#include <cstring>
#include <iostream>

using namespace std;
main()
{
    int iLen;              //定义变量，用来存放字符串长度
    char    *strF;         //定义第一个字符串变量
    char    *strS;         //定义第二个字符串变量
    char    *strT=new char[20]; //定义第三个变量并分配空间（字符数组）

    //为第一个和第二个字符串赋值
    strF="just try,";
    strS="sigh!";

    //利用库函数，将第一个字符串复制到第三个字符串
    strcpy(strT, strF);

    //将第二个字符串连接到第三个字符串上
    strcat(strT, strS);

    iLen=strlen(strT);

    cout<<strT<<","<<iLen<<endl;

    delete  [] strT;          //释放分配的内存空间
    return 0;
}
```

这段代码很好懂，只是简单地将字符串互相连接并且输出，其运行结果非常简单，如图 1-1 所示。

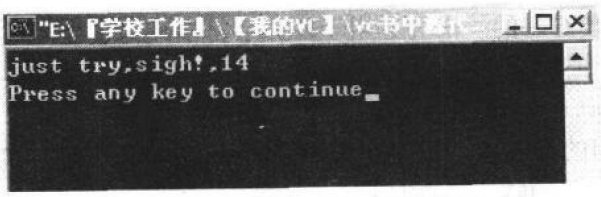


图 1-1 C 风格字符串的一个演示窗口

写出这么一段简单代码，正是为了对 C 风格的字符串进行一些简单的分析：首先是定义 3 个字符串，其中，字符串 strT 在定义时为其分配存储空间。如果不分配存储空间，那么，在

下面的 strcat()函数执行的过程中，将会出现错误。需要注意的是，在分配了存储空间之后，要记得使用 delete 语句收回不再使用的空间。

在定义完字符串后，利用 strcpy()和 strcat()函数来实现字符串的连接。这两个函数搭配用来实现字符串的连接。其中 strcpy()用来将字符串复制为自身的一个新的实例，然后把另一个新的字符串和这个新的实例进行连接。这几步的操作如果不恰当，将会造成内存错误。在这里值得说明的是，不能像 VB 和 Delphi 那样，直接使用“+”来连接两个字符串，而必须利用库函数来进行。

C 风格的字符串涉及了大量的指针运算，因此在编程中容易出现错误。作者建议，尽量少用 C 风格的字符串，而使用下面将要讲述的标准 C++的 string 类型。

1.1.2　string 类型字符串

C 风格字符串主要使用字符指针来进行操纵，因而容易出现错误。为了避免这些“使用 C 风格字符串相关的错误”，许多开发部门、组织都提供了自己的字符串类（在 Visual C++中，就提供了 CString 类，用来实现字符串的操作）。这些字符串类的不同实现方式，使得程序的兼容性和可移植性变得非常困难，针对这种情况，标准 C++提供了 string 类，用来支持字符串的操作。要使用 string 类，需要包含相关的头文件：

```
#include <string>
```

标准 C++中的 string 类提供了针对字符串的一系列操作，包括字符串的比较、连接、长度查询、字符串之间的拷贝等。另外，标准 C++的 string 类型支持泛型运算，能够提高编程的效率和程序的可移植性。

下面是一段利用 string 类编写的字符串操作的代码：

程序清单 1-2：

```
#include <iostream>
#include <string>
#include <algorithm>     //加入泛型算法的支持

using namespace std;

void main()
{
    int iLen;          //定义存放字符串长度的变量
    string  strF;          //定义第一个字符串
    string  strS;          //定义第二个字符串
    string  strT;          //定义第三个字符串

    strF ="Just a Standard C++ String!";
    strS ="The Second string.";
    strT =strF + strS;  //比较好的字符串连接方式
    iLen =strT.length();

    cout<<strT<<" ,  "<<iLen<<endl;
```

```
    //下面是 C 风格字符串和 string 类型字符串的转换问题
    char    *cStr;
    string  strStd;

    cStr ="This is C Style!";
    strStd =cStr;          //可以直接将 C 风格的字符串转换为 string 类型字符串

    cout<<endl<<strStd<<endl;

    //cStr =strStd; 这条语句是错误的，不能直接将 string 类型的字符串
    //转换为 C 风格的字符串，正确的写法如下：

    strStd ="将 string 字符串转换为 C 风格字符串！ ";

    const char *cStrSec =strStd.c_str();     //初始化一个 C 风格字符串，
                                             //并用 string 字符串为其赋值
    cout<<endl<<cStrSec<<endl;

    //string 类型的字符串也可以利用下标来进行访问单个的字符
    strStd ="Just a string test!";
    cout<<endl<<"输出 string 类型字符串的第三个字符:"<<strStd[2]<<endl<<endl;

    //测试一个泛型算法
    replace(strStd.begin(), strStd.end(), 's', 'M');
    cout<<strStd<<endl<<endl;
}
```

这个简单的例子，用以演示标准 C++字符串的一些基本功能。包括了字符串的连接、不同风格字符串之间的互相转换、字符串的替代等。另外，为了使用函数 replace，需要包含头文件<algorithm>，用以支持泛型算法。需要说明的是，这段代码的编程风格不是很好（由于需要讲述几个问题，因而生硬地将程序分成了几段；另外，在用到某个变量时才开始定义该变量），只是为了讲述语言特性而用。

上面例子的运行结果如图 1-2 所示。

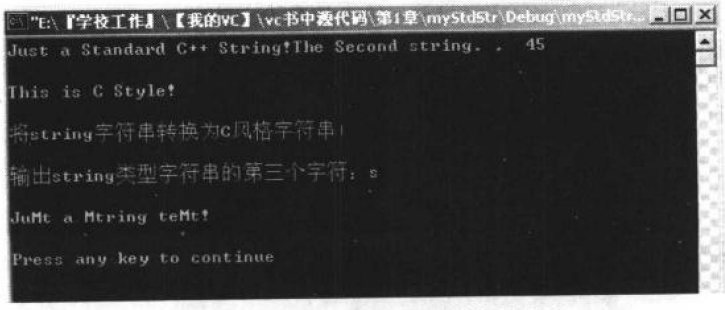


图 1-2 string 类型字符串的简单演示

由上面的例子容易看出，string 类型的字符串便于操作，而且支持泛型算法，这使得对字符串的操作变得容易而灵活。另外，string 类型的字符串还有更多的用法，在此不再赘述。

虽然 string 类型的字符串是推荐使用的操作字符串的方法，但是 C 风格的字符串在许多应用程序中也有着广泛的应用，因而仍然需要对其进行比较深入的学习。

1.2　指针与引用

指针一直是 C/C++语言中的难点，它在带来了灵活性的同时也增加了程序出错的可能性。然而，毫无疑问，指针有着非常强大的功能。合理、正确地使用指针，将会使程序更加灵活而高效。本节将针对指针使用中需要注意的一些问题进行讲述。

1.2.1　指针的算术运算

首先，看一个关于指针的程序片断：

```
#include <iostream>

using namespace std;
void main()
{
    int iVal=5;
    int *pi = &iVal;     //如果声明为 int *pi = iVal 则出现错误

    cout<<*pi<<endl;
    cout<<pi<<endl;
}
```

程序运行后，首先输出*pi 的值，它等于 iVal，即 5；下一行，程序输出 iVal 所在的内存地址，作者机器上的运行结果为 0012FF7C。

如注释所显示，如果要初始化指针，则需要为其赋值为一个地址，而不是变量的值。指针的复杂性表现于其双重性质。

pi 表示指针内含的内存地址；*pi 则表示指针所指的对象值。关于这点一定要注意区分。另外，指针可能会不指向任何对象。此时，指针内含地址为 0，即空指针（null 指针）。如果访问一个空指针，将会出现不可预测的结果，造成内存访问错误。

对于指针，可以执行算术运算，使得其地址值增加或减少一个整数。这样的指针操作即为指针的算术运算。对指针进行算术运算，等于为指针内含的地址增加该指针所指对象的长度的倍数的大小。如果对于整型指针加 1，如上面的例子，那么，其地址实际上加的是 8（因为 int 类型的长度为 4 个字节）。

当指针指向数组元素时，才能确保指针算术运算能够很好地应用。因为数组中的元素是连续存储在内存之中的，而逐个声明的变量并不一定能确保其在内存地址中是连续的。指针算术

运算最典型的用法就是遍历数组中的每个元素。

在本小节中，需要记住：使用指针前，要检验其是否为空；在使用指针算术运算时，要确保其能够指向有效的内存地址。

1.2.2 加上 const 修饰符的指针

有 const 修饰符的指针随着 const 出现的位置的不同而有不同的含义。这是一个容易混淆的地方，同时也是一个非常重要的地方，有必要专门对其进行讲述。

1. 指向常量的指针

首先，声明一个指针，如下所示：

```
const  double  *pDbl;
```

此时，pDbl 是一个指向 double 类型的 const 对象的指针。其含义简单解释就是：pDbl 本身不是 const 类型，它是可变的，可以通过重新赋值来指向不同的对象；而 pDbl 所指向的对象*pDbl 则是 const 类型，是不可变的，即不能对*pDbl 赋值。

下面给出一个程序片断，其中的注释文字解释了一些关键的问题。

```
//指向 const 对象的指针
double  dblVal =5.0;
double  dblTry =6.0;
const   double   *pDbl =0; //声明指向 const 对象的指针

pDbl =&dblVal;  //将变量 dblVal 的地址赋给指针
//*pDbl =7.0;   这条语句会出错，因为，*pDbl 对应的是常量。
//然而，语句 dblVal =7.0;却不会错，此时，*pDbl 的值也变为 7.0。
//因此，为了确保不能改变 dblVal 的值，应该声明 dblVal 为如下语句：
//const dblVal =5.0;
cout<<*pDbl<<endl;
pDbl =&dblTry; //pDbl 是可变的，为其赋不同的地址值，使其指向不同的对象
cout<<*pDbl<<endl;
```

上面的代码片断演示了关于指向常量的指针的一些细节问题。在实际的应用中，指向常量的指针常常被用作函数的形式参数。例如：

```
int OpenFile(const char *strFileName);
```

需要注意的是如下写法也表示指向常量的指针：

```
double const *ptr =&dblF;
```

2. 常量指针

常量指针，其所指向的对象是可变的；而指针内含的地址则是不可变的。示例代码片断如下：

```
//const 指针
double dblF =7.0;
double  dblS =8.0;
double *const ptr =&dblF;//声明常量指针
```

```
//ptr =&dblS;这条语句错误，此时的指针是常量，不能为改变指针的内含地址值
cout<<*ptr<<endl;

*ptr =dblS;//常量指针，然而其所指的对象却是可改变的
cout<<*ptr<<endl;
```

可以看出，const 修饰符的位置不同，指针的含义和效果就不相同。如果想要声明一个常量指针，使其指向 const 变量，那么需要按如下语句进行声明：

```
const double *const pCtr =&dblF;//声明常量指针，指向常量对象
```

由上面的例子可知，定义一个常量指针或指向常量对象的指针一般都要写比较长的语句。为了简化代码，可以利用 typedef 关键字来定义类型的别名，示例如下：

```
typedef const double *pc;//定义指向常量对象的指针
typedef double *const cp;//定义常量指针
```

这样，就定义了 pc 类型，用来声明指向常量的指针；定义了 cp 类型，用来声明常量指针。用法如下：

```
cp p1 =0;    //此时 p1 是常量指针
pc p1 =0;    //此时 p1 是指向常量对象的指针
```

可见，利用 typedef 关键字可以简化代码，而且可以为常用的数据类型定义直观简洁的别名，从而增加程序的可读性。在 Windows API 函数中以及微软的 MFC 类库中，就有不少利用 typedef 关键字定义的数据类型。

注意：使用 typedef 关键字时，有一个地方需要注意。如下定义：

```
typedef double * pDbl;
```

这个声明其实是定义了一个指针类型 pDbl，如果此时使用 const 修饰符声明如下：

```
const pDbl dblQ =&a1;
```

那么，这个声明表示的意思是：

```
double * const dblQ;
```

而不是：

```
const double * dblQ;
```

1.2.3　函数指针

函数指针能够作为函数的参数，这对于撰写通用性强的代码有重要的意义。函数指针的具体声明方法如下：

```
TypeName  (*FuncName)(Param List);
```

下面的例子即为声明了一个返回值为 double 类型的函数指针：

```
double (*pF)(int, int);
```

函数指针在使用中需要注意以下几个方面：

- 上面的式子如果写为如下形式：

```
double *pF(int, int);
```

则不能代表一个函数指针，而表示一个函数 pF(int, int)的返回值是一个 double 类型。