引　言

面向对象方法和 Windows 环境是当今计算机领域的热门话题。作为 90 年代软件开发的主流，面向对象方法集成和协调了过去几十年软件开发实践的精髓，支持从对象、类、类库、直至专用系统构架的多层次抽象，为大规模软件重用和复杂工程软件开发提供了有力工具。而 Windows 环境实际上已成为台式操作系统的事实标准，正逐步取代 DOS 的地位，并向工作站和高档服务器渗透。由于 C++ 同时支持面向对象软件开发方法和 Windows 开发环境，近年来发展十分迅速，已在许多行业得到了广泛的应用，成为开发大型工程应用软件的首选语言。

软件重用是提高软件开发效率和质量的重要途径，C++ 通过类库支持软件重用。开发高质量可重用 C++ 类库是实现大规模软件重用、避免低水平重复劳动的前提条件。本书介绍的 MATCLIB 是作者近期完成的一套可重用矩阵运算类库，它利用面向对象思想和 Borland C++ 功能强大的类模板特性开发，就象一块软件“IC”，可方便地用于以C++为开发语言的科技应用软件中。

0.1　本书的特点

目前，虽然有关面向对象编程(OOP)、C++、Windows 的书籍随处可见，但大多数侧重介绍概念、语言和开发环境，从工程应用软件和实际开发经验角度进行阐述的作品很少。对于已具备 C++ 和 Windows 基本知识的读者，可重用软件模块与实用开发经验常常更具有现实意义和应用价值。本书是作者针对这种状况推出的一部软件开发专著，它立足于当今最流行的 Windows 环境，通过分析一套完整的应用软件(MATCLIB)来阐述 C++ 面向对象特性的应用技术。本书以科学性、新颖性、实用性为目标，反映了作者近几年的最新研究成果，旨在为正在学习或使用 C++ 的软件开发人员提供源于实践的C++和 Windows 应用软件开发经验，以及一套可参照、修改、扩充、重用的工程软件系统。

MATCLIB 在逻辑结构上可分为两个层次：完成向量矩阵内存管理和基本运算的内核层，以及定义常用向量矩阵函数的扩充层，扩充层中的函数用户可根据实际需要自行裁剪或增添。目前，MATCLIB 实现了通用向量函数近百种，矩阵函数约二百种。可用于整型、单精度实型、双精度实型、复数型向量和矩阵的辅助运算，结构精巧，功能强大，使用方便、易于重用和扩充。MATCLIB 的主要特点如下：

- 用算术操作符完成向量和矩阵的基本运算；
- 自动进行下标越界检查；
- 可进行特大维数的矩阵定义和运算(单个矩阵容许内存 1GB)；
- 与 C++ 数组完全一致的操作风格；
- 动态建立向量和矩阵对象；

· 流式输入/输出向量和矩阵；

· 自动报告操作错误；

· 灵活的向量和矩阵生成方式；

· 同时适用于 DOS 和 Windows 环境；

· MATCLIB 中的矩阵(向量)对象可作为实参调用以 C++ 二维(一维)数组作形参的函数；

· 支持用户扩充。

MATCLIB 提供三种形式的类库：DOS 静态连接库、Windows 静态连接库以及 Windows 动态连接库，适用于 DOS 和 Windows 两种开发环境以及不同层次的科技应用软件开发需求。在 MATCLIB 中，涉及到 C++ 软件开发的各个方面，从继承、重载、到多态、模板，集中反映了 OOP 方法的新特点。

本书是作者近几年 OOP 软件开发经验的结晶，类库中用到的许多 C++ 编程技巧都是作者通过大量的编程实践获得的，具有极大的实用价值。MATCLIB 中的核心操作和常用函数都作过精心设计，以满足科学计算对运行效率的严格要求。本书中给出的相关代码全部从 MATCLIB 源文件中摘录而来，确保正确无误。书中的示例程序均在本书配套软盘中提供，可调入工程文件编译运行。配套盘中还装有 MATCLIB 的全套实现源代码和类库软件，能直接用于开发用户自己的科学计算和工程应用软件，提高用户的软件开发效率。因而，读者从本书不但可以学习 C++ 面向对象软件的设计思想和实现方法，掌握 Windows 环境下应用软件的基本开发技巧，快速获得 C++ 软件实际开发经验，还可直接得到一套可重用的科学计算类库软件。

0.2 本书的基本组成

本书对 MATCLIB 作了全面剖析，首先向读者描述 MATCLIB 的各种独特性能，再逐一介绍这些功能的实现途径，并用实例演示了它们的使用方法。由于 MATCLIB 涉及到 Windows 环境，因而对 Windows 应用软件开发方法作了简要介绍，限于篇幅，本书只介绍了基于 OWL(Object Windows Library)的常用接口对象设计方法，并提供了相关示例程序。对 Windows 软件开发方法有兴趣的读者还可参阅下述文献：

[1] Microsoft Corporation 编，宋明华 童长忠 任洪江译，Windows V3.0 程序设计指南和工具”，电子工业出版社，1991

[2] 建平 威文 晓林 叶舟编译，Microsoft Windows 3.0 程序设计与实例，北京科海培训中心

[3] 马强 柯源 吕永奇编译，Borland C++ 环境下的 Windows 编程技术与实例，海洋出版社，1992

[4] [美] Ted Faison 著，蒋维杜 吴志美 张新宇 李景淑译，Borland C++ 3.1 编程指南，清华大学出版社，1993

文献[1]和[2]循序渐进地介绍了 Windows 应用软件的基本结构和各种接口对象设计方法，特别适合于缺乏 Windows 开发经验的读者使用。

文献[3]和[4]对基于 OWL 的应用软件开发方法的介绍各具特色，适用于具有一定 Windows 软件开发经验的读者。

本书共分为 10 章，引言部分描述本书的基本结构和对软硬件的要求；第一章综合介绍 MATCLIB 类库的基本特色，包括它的设计思想、技术特点和一个反映 MATCLIB 基本用法的简单示例；第二章结合 MATCLIB 讨论 C++ 类模板的使用技巧，并描述了 MATCLIB 中用到的全程变量和错误报告类；第三到第八章详细分析 MATCLIB 的结构和功能，以及在 DOS 环境下的使用方法；第九章介绍基于 OWL 的应用程序结构和常用 Windows 接口对象设计，它为理解第十章的内容奠定基础；第十章分析 MATCLIB 在 Windows 环境下的应用实例以及 MATCLIB 动态连接库的设计和使用技巧；附录 A 提供了 MATCLIB 字符串类的实现代码；附录 B 中列出了部分篇幅较长的矩阵扩充函数；附录 C 为 MATCLIB 主要操作和函数速查表，便于用户使用时查阅。

0.3 本书的软硬件需求

为了掌握本书的内容和运行书中的示例程序，需要下述软硬件的支持：

- 386 以上的微机
- 与 Microsoft 兼容的鼠标
- VGA 或更好的图形适配器
- Borland C++ 3.1 及其应用框架
- MS-DOS 5.0 或更高版本
- MS Windows 3.1 或更高版本

虽然原则上可在各种内存模式下建立 MATCLIB 类库，但由于 MATCLIB 规模较大，仅提供单一代码段(Small, Compact)或单一数据段(Small, Medium)的内存模式并不实用，因此建议读者使用 Large 模式编译。本书提供的所有工程文件均设置为 Large 模式，MATCLIB 类库也在该模式下建立。由于 C++ 标准头文件和库文件在每台计算机中的安装位置均不相同，读者在使用本书配套中的工程文件时，需要重新设置路径，以便编译及连接系统能正确定位头文件和库文件。

第一章　MATCLIB 的特点

C/C++ 语言具有极好的灵活性和丰富的功能，“入门容易得道难”非常形象地描述了学习 C/C++ 语言的基本特点。掌握 C/C++ 的程序结构和语法只是入门而已，只有用它解决实际问题，在编程实践中不断探索和总结各种解决问题的方法和技巧，才能获得其精髓，发挥出它的极大潜力。通过编程学习语言是快速掌握算法语言的捷径，而从别人的程序中获取特殊技巧和方法，则是迅速提高软件开发水平的秘诀，因为它们常常是编写者经过艰苦摸索得到的宝贵经验，从现成文档中是无法得到的。本书希望通过对 MATCLIB 的剖析，向读者提供源于实践的 C++ 软件开发经验，帮助读者迅速掌握 C++ 各种新特性，如继承、重载、多态、模板等的实际使用技巧，并学会科技应用软件中常用 Windows 接口对象的开发方法。

本章首先概要介绍 MATCLIB 的外部特征，包括它的设计思想、技术特点和应用程序结构，提供一个 MATCLIB 的总体轮廓。

1.1　MATCLIB 的设计思想

向量与矩阵运算在科学计算软件中占有很大比重，对科技软件的开发效率和质量起着举足轻重的作用。C++ 语言以数组表征向量和矩阵，并提供了灵活的指针来协助数组的管理和操作，使参数传递简单、迅速。但由于 C++ 数组不能跟踪它所包含的元素数目，很容易在毫无知觉的情况下对数组作越界操作，导致莫名其妙的运行失败或死机现象，因而，使用 C++ 数组具有很大风险性。除此之外，数组操作也很不方便，一个简单的二维数组赋值功能也需要编写好几行语句。虽然在实际的软件开发中，程序员一般并不编写这些通用数组操作函数，而是借助于标准数学库和自己开发的各种函数库来完成相应功能，但这并未使程序员从困境中摆脱出来，其根本原因在于调用一个函数时需要程序员了解和传送的信息数量太多，既繁琐又容易出错。以二矩阵相乘为例，其典型函数原型格式如下：

```
void matmult(float **m1, float **m2, int l, int m, int n, float **m3);
```

如果没有对上述函数各变量的详细说明，要使用它是不可能的。即使已确知各变量的具体含义，要保证在调用过程中正确传递各参数也不是一件容易的事。假如能找到一种合适的途径，可以对应用软件开发者隐藏大多数细节因素，能象标量相乘一样通过 m1 * m2 来计算二矩阵的乘积，无疑可大大减轻开发者的编程负担，提高软件的可读性和可维护性。这正是我们当初研究 MATCLIB 的基本设想。

实现上述设想需要新的软件开发方法和工具，近年来，面向对象编程(OOP)技术得到了很大发展，它所具有的信息隐蔽、数据抽象、继承、动态联编等特点可显著降低软件的复杂性，在人机界面、数据库、多媒体等领域获得了广泛应用。有的人认为，用 OOP 开发的

软件运行效率低，不适合于科学计算。实际上 OOP 与低效率之间并无必然联系，关键在于对语言的选择和 OOP 特性的合理运用。我们通过对比计算发现：如果选用 C++ 语言，并适当控制使用迟后联编(动态联编)，具有典型 OOP 特点的 C++ 软件与用常规 C 语言开发的软件运行速度相当，但它在简化程序接口，降低软件复杂性方面的优势则是后者无法比拟的。因而，C++ 语言同时兼备 OOP 语言的灵活性和常规语言的效率，是开发科技软件的理想编程语言。我们开发 MATCLIB 时充分利用了 Borland C++ 3.1 的面向对象特性，包括继承、重载、多态性、类模板等。

MATCLIB 以类库形式组织，包含万余行 C++ 源代码。对向量而言，定义有适合于整型、单精度实型、双精度实型、复数型向量的通用向量函数约 80 个，专用于双精度实型、复数型向量的函数各 30 余个。矩阵运算方面，定义有适合于整型、单精度实型、双精度实型、复数型矩阵的通用函数 100 余个，专用于双精度实型矩阵的函数约 60 个，专用于复数型矩阵的函数 30 余个，以及部分类型转换函数。

在 MATCLIB 中，向量和矩阵作为能动对象知道自身的结构，因而可对外界隐藏许多实现细节，大大简化了用户接口，降低了软件重用难度。用 MATCLIB 开发的软件简洁直观，易于维护。借助于 MATCLIB，矩阵运算的编程工作变得非常简单。例如，假设 a, b, c 为维数相容的已知双精度实矩阵，我们要将 a 和 b 相乘，对乘得的结果求逆，再与 c 相加，最后计算前面结果的行列式，这一系列运算可简单地用下面一条语句描述：

```
result = det(inv(a * b) + c);
```

result 是存放结果的双精度实数。不难想象，如果采用传统的子程序调用方式，至少需要四条调用语句，形式要复杂得多，而且需要程序员定义临时矩阵存放中间结果。一种可能的形式如下：

```
matprod(a, b, n, m, n, temp1);  //调用矩阵相乘函数
inv(temp1, n, temp2);           //调用矩阵求逆函数
matadd(temp2, c, n, n, temp3);  //调用矩阵相加函数
result = det(temp3, n);         //调用行列式计算函数
```

其中 temp1, temp2, temp3 是用于存放中间结果的临时矩阵。

由此可见，MATCLIB 对简化向量和矩阵编程、提高科学计算软件开发效率具有显著效果。

1.2 MATCLIB 的技术特点

MATCLIB 类库以 Borland C++ 的类模板特性为基础，主要由下述 14 个类构成：

String	通用字符串处理类；
ErrorReporterbase	错误报告基类；
DosErrorReporter	DOS 环境错误报告类；
WinErrorReporter	Windows 环境错误报告类；
vector<Type>	通用向量模板基类；
ivector	整型向量类，与 vector<int>相同；

fvector	单精度实型向量类，与 vector＜float＞相同；
dvector	双精度实型向量类，它在 vector＜double＞基础上扩充；
cvector	复数型向量类，它在 vector＜complex＞基础上扩充；
matrix＜Type＞	通用矩阵模板基类；
imatrix	整型矩阵类，与 matrix＜int＞相同；
fmatrix；	单精度实型矩阵类，与 matrix＜float＞相同；
dmatrix	双精度实型矩阵类，它在 matrix＜double＞基础上扩充；
cmatrix	复数型矩阵类，它在 matrix＜complex＞基础上扩充。

其中：DosErrorReporter 和 WinErrorReporter 是 ErrorReporterbase 的导出类；dvector 和 cvector 分别为 vector＜double＞及 vector＜complex＞的导出类；而 dmatrix 与 cmatrix 则分别是 matrix＜double＞与 matrix＜complex＞的导出类。

MATCLIB 是对常规 C/C++ 数组的面向对象扩展，它具有许多新的特点，现描述如下：

(1) 采用动态内存分配技术建立任何向量和矩阵对象。对局部向量或矩阵，超出作用域时由析构函数自动回收其占用的内存。

(2) 用算术操作符完成向量和矩阵的基本运算。在 MATCLIB 中，各类对象间的合法基本运算均可用简洁易读的算术运算符来表达，这些操作符除适用于同类对象间的运算外，还可用于各类对象与相应基本数据类型(int, float, double, complex)间的运算。此外，还定义了对不同类型的向量或矩阵作相容转换的函数，可方便地将整型、单精度实型向量和矩阵转换为双精度实型，或将双精度实型向量和矩阵转换为复数型。如果 cx、cy、cresult 为复数矩阵，dz 为双精度实矩阵，cconst 为一标量复数，下面的语句将是合法的：

```
cresult = cx * cy - Complex(dz) + cconst;
```

它将计算 cx 与 cy 的乘积(维数应相容)，然后与由 dz 转换来的复数矩阵的各对应元素相减，再将复数标量 cconst 加到每个元素上，运算结果存于复数矩阵 cresult 中。

(3) 自动进行下标越界检查。当用超过向量或矩阵定义范围的下标检索其元素时，将报告一个致命错误，以防止对超出边界以外的元素进行访问。

(4) 可进行特大维数的矩阵运算。在内存容量许可的条件下(Windows 环境下由于采用磁盘对换技术，可访问多达 4GB 的虚拟内存)，可随意定义向量或矩阵，只要单个向量和矩阵的维数不超过下述限制即可：

ivector (32752)	imatrix (16382 * 32752)
fvector (16376)	fmatrix (16382 * 16376)
dvector (8188)	dmatrix (16382 * 8188)
cvector (4094)	cmatrix (16382 * 4094)

从技术上讲，采用巨型指针可容易地消除上述维数限制(此时整型向量可定义到 2147483647 维，其他向量和矩阵的最大维数参照上述关系容易得出)，但我们并没有这样作，主要出于对运行效率的考虑。巨型指针需要对 32 位地址进行计算，将大幅度降低向量和矩阵的访问速度，而科学计算软件中速度常是一个至关重要的因素。此外，就当前计算机的内存配置而言，上述维数范围已完全可以满足绝大多数科技软件开发需要，因为一

个维数为 16382 * 8188 的双精度矩阵，其所占用内存已高达 16382 * 8188 * 8 = 1073MB 或者 1GB，在目前的科学计算中，单个矩阵所占内存超过 1GB 的情况是极其罕见的。

(5) 与 C++ 数组完全一致的操作风格。对 MATCLIB 中的向量和矩阵，可使用与常规动态数组相同的操作方法。向量对象能用于各种使用 C++ 一维动态数组的地方，包括调用以一维数组指针为形参的常规函数，不需要程序员作任何中间转换。而 MATCLIB 中的矩阵则可用于各种使用 C++ 二维动态数组的场合，可直接作为实参调用以二维数组指针为形参的常规函数。由于 MATCLIB 实际上是对 C++ 标准运行时刻库中数组部分的扩充，它兼容各种用标准 C/C++ 编写的主程序或函数。

(6) 流式输入和输出向量与矩阵。MATCLIB 中定义的各型向量和矩阵均可直接连接到控制台、内存缓冲区、或磁盘文件上实现流式输入或输出操作，使用非常方便。以下是一个输出 2 * 3 随机矩阵 xm 的例子：

```
控制台输出：
    cout<<"xm = \n"<<xm<<"\n";
磁盘文件输出：
    ofstream outf("xm.out");        //创建流式输出文件"xm.out"
    if(outf != NULL)
    {                               //若创建成功，执行输出操作，然后关闭文件
        outf<<"xm = \n"<<xm<<"\n";
        outf.close()
    }
它们得到的输出格式将如下所示：
xm =
[[0.010559,0.003967,0.335154]
 [0.033265,0.355724,0.2172]]
```

(7) 优良的扩充性能。用户不仅可将 MATCLIB 用作一个类库调用其中定义的函数和操作，还可用作一个可扩充的矩阵运算平台随时并入新的功能或对一些自己使用较少的功能进行裁剪。扩充方式有两种：直接在原类中增加新的成员函数，或在继承原类基础上定义一新的类，在新类中定义扩充方法。前种方式要修改原来的类定义，可能产生副作用影响原类库的性能，一般仅供类库开发者使用。后种方式通过继承获得原来类中定义的各种函数和操作，在此基础上定义扩充方法。由于它不对原类库作任何改动，没有副作用，不需要原类库的实现代码便可进行，特别适合于类库用户。对每种扩充方式，均可采用下面两种途径实现新定义的方法：

I：为已有的常规 C/C++ 子例程编写一个简单的包装函数作为类的成员或友元。假如我们已用常规 C 语言编写了一个计算双精度实数矩阵逆阵的子例程 matinvert，其原型如下：

```
void matinvert(double **a,int numrow,double **inva);
```

则可将包装函数写为：

```
dmatrix inv(dmatrix &a)
{
    int numcol = a.mrow();          //检索矩阵 a 的维数
```

```
    dmatrix inva(numcol,numcol);        //定义一矩阵用于存放逆阵
    matinvert(a,numcol,inva);           //调用常规例程
    return inva;                        //返回逆阵
}
```

容易看到,包装函数的唯一核心语句便是以矩阵作实参调用我们原来定义的常规子例程,这样,不需要对 matinvert 作任何改动便可将它以 inv 函数形式并入 MATCLIB 库中,和 MATCLIB 的其它函数享有同样的灵活方便性。这种方法对高效发挥已有软件资源的作用具有特别重要的意义。

II:利用 MATCLIB 中已定义的函数或操作实现扩充方法。它常能用很少的语句完成很复杂的计算。例如,要扩充一个求线性代数方程组最小二乘解的方法,利用 MATCLIB 中已有的计算广义逆函数 ginv 和矩阵与向量相乘算子"*",只需一条计算语句即可实现,具体形式如下(假设扩充的是友元函数):

```
dvector cmgm(dmatrix & a,dvector & b)
{
    int n = a.mcol();               //检索矩阵 a 的列数
    dvector result(n);              //定义一向量用于存放解向量
    result = ginv(a) * b;           //计算解向量
    return result;                  //返回解向量
}
```

其中 a 为方程组系数矩阵,b 为右向量,result 为方程组的解。

(8) 完善的错误报告机制。MATCLIB 的错误报告提供下述信息:错误的性质及其简要说明;发生错误的文件名及出错的行号。

错误性质分为两大类:警告错误(Warning Error)和致命错误(Fatal Error)。警告错误提醒用户 MATCLIB 当时提供的结果不太可靠,比如,对接近奇异但尚不满足奇异条件的矩阵求逆阵等,它一般不影响程序继续执行。致命错误则表示一个严重影响甚至可能破坏应用程序的错误已经发生,必须立即终止应用程序,如内存不够、二相乘的矩阵维数不匹配等。如果程序运行于 DOS 环境,MATCLIB 遇到致命错误时将显示错误信息并终止程序,遇到警告错误时显示错误信息,但程序继续运行。如果程序运行于 Windows 环境,遇到错误(包括警告或致命错误)时将弹出一信息框提供出错信息,对警告和致命错误的后端处理与 DOS 环境类似。

(9) 灵活多样的向量和矩阵生成方式。对实型向量和矩阵均提供了七种构造方式,以满足不同需要。简单示例如下:

```
double * f = new double[4];     //定义一个普通的 C++ 一维数组
......
dvector vv;                     //缺省构造函数,生成一个所含元素数为 0 的空向量

dvector av(10);                 //构造有 10 个元素的向量,元素初值缺省为 0
dvector bv(5,1.2);              //构造有 5 个元素的向量,元素初值均为 1.2
dvector cv(f,4);                //用指向普通 C/C++ 一维数组的指针构造向量
dvector dv(av,bv);              //连接向量 av 和 bv 构造一新向量
```

```
dvector ev(dv);                    //生成一个与原向量 dv 内容相同的新向量
dvector fv(dv,3,11);               //构造向量 dv 的一个子向量,有 11-3=8 个元素

double **f=new double *[4];        //定义一个普通 C++ 二维数组
for(int i=0; i<4; i++)
  f[i]=new double[5];
......
dmatrix mm;                        //缺省构造函数,生成一个行列数均为 0 的空矩阵

dmatrix am(10,8);                  //构造 10*8 矩阵,各元素初值缺省为 0
dmatrix bm(5,5,1.2);               //构造 5*5 矩阵,各元素初值均为 1.2
dmatrix cm(f,4,5);                 //用指向普通 C/C++ 二维数组的指针构造矩阵
dmatrix dm(am,bm);                 //将两个矩阵 am 和 bm 组合成一新矩阵(按列组合)
dmatrix em(dm);                    //生成一个与原矩阵 dm 内容相同的新矩阵
dmatrix fm(dm,3,11,1,6);           //构造矩阵 dm 的一个子矩阵,其
                                   //维数是(11-3)*(6-1)=8*5
```

复数型向量和矩阵还提供了另一种形式的构造函数,它以两个双精度实型向量或矩阵为参量:

```
int n,m;
......
dvector v1(n),v2(n);
......
cvector vv(v1,v2);     //构造一个复向量,它分别以 v1、v2 为实部和虚部

......
dmatrix m1(m),m2(m);
......
cmatrix mm(m1,m2);     //构造一个复矩阵,它分别以 m1、m2 为实部和虚部
```

(10) 适用于 DOS 和 Windows 两种开发环境和不同层次的需求。对类库中不兼容于两种环境的操作(如控制台输入输出),MATCLIB 依靠编译时的标准宏定义选取不同的处理方法。为满足不同应用软件的需要,MATCLIB 提供了两大类共五种类型的矩阵库:

标准库:它拥有本书将介绍的各种向量和矩阵运算方法,它有三种类型:

a MATLIB.LIB: DOS 静态连接库;

b MATLIBW.LIB: Windows 静态连接库;

c MATDLL.DLL, MATDLL.LIB: Windows 动态连接库与输入库。

简缩库:它提供向量和矩阵的基本操作和函数,是通过裁减标准库得到的,主要目的是压缩库文件规模,为 DOS 下的应用软件留下更多的内存空间,简缩库有两种类型:

a MATPAC.LIB: DOS 静态连接库;

b MATPACW.LIB: Windows 静态连接库。

对 DOS 环境下的应用软件,若不需要范数、条件数、奇异值分解、特征值等高级函数,建议使用简缩库,它可压缩运行文件规模。对 Windows 环境下的应用软件,若用到该环境的多任务特性,采用动态连接库可显著减小内存开销,并能缩小运行文件尺寸。如果用

户要经常用到一些 MATCLIB 中未有的新的向量或矩阵函数，可按本节(7)介绍的方法自行扩充。

MATCLIB 各版本涉及到的主要源文件共有 32 个，各文件的名称及主要作用列于表 1.1 中。

表 1.1　MATCLIB 源文件一览表

文件名称	主要内容	文件名称	主要内容
MATDEF.HPP	宏定义、全程变量	MATERR.HPP	错误信息
STR.HPP	String 类定义	STR.CPP	String 类实现
ERCLASS.HPP	错误报告类定义	ERCLASS.CPP	错误报告类实现
BVECLIB.HPP	vector 类模板定义	BVECLIB.CPP	vector 类模板实现
BMATLIB.HPP	matrix 类模板定义	BMATLIB.CPP	matrix 类模板实现
VECLIB.HPP	各向量导出类定义	VECLIB.CPP	各向量导出类实现
MATLIB.HPP	各矩阵导出类定义	MATLIB.CPP	各矩阵导出类实现
BVECPAC.HPP	vector 定义(简缩版)	BVECPAC.CPP	vector 实现(简缩版)
BMATPAC.HPP	matrix 定义(简缩版)	BMATPAC.CPP	matrix 实现(简缩版)
VECPAC.HPP	向量类定义(简缩版)	VECPAC.CPP	向量类实现(简缩版)
MATPAC.HPP	矩阵类定义(简缩版)	MATPAC.CPP	矩阵类实现(简缩版)
BVECDLL.HPP	vector 定义(DLL 版)	BVECDLL.CPP	vector 实现(DLL 版)
BMATDLL.HPP	matrix 定义(DLL 版)	BMATDLL.CPP	matrix 实现(DLL 版)
VECDLL.HPP	向量类定义(DLL 版)	VECDLL.CPP	向量类实现(DLL 版)
MATDLL.HPP	矩阵类定义(DLL 版)	MATDLL.CPP	矩阵类实现(DLL 版)
MATSUB.HPP	矩阵高级函数原型	MATSUB.CPP	矩阵高级函数源代码

1.3　一个简单的示例程序

以上对 MATCLIB 的特点作了说明，为了提供一个更完整的感性认识，本节特用 MATCLIB 编写了一个示例程序，该程序计算随机矩阵 s 的广义逆 s^+，以及另一随机矩阵 a 的特征值 d 和特征向量 v，并用下述公式验算所求结果：

$(s^+)^+ = s$,　　$a * v = v * d$。

完整程序见示例 1.1，由于程序中大多数语句后面均有说明，故不再作更多解释。

示例 1.1　基于 MATCLIB 的应用程序结构

```
#include "matlib.hpp"          //头文件,使用 MATCLIB 标准静态库的程序所必需
int main(void)
{
```

```
  int n1,m1,n2;
  cout<<"input n1,m1(the dimension of matrix s):\n";  //提示输入
  cin>>n1>>m1;                                        //输入矩阵 s 的阶次
  cout<<"input n2(the dimension of matrix a):\n";     //提示输入
  cin>>n2;                                            //输入方阵 a 的阶次
  dmatrix s(n1,m1),vs(m1,n1),vvs(n1,m1),a(n2,n2);     //构造动态实矩阵
  cmatrix d(n2,n2),v(n2,n2),av(n2,n2),vd(n2,n2);      //构造动态复矩阵
  s.rand();                                  //将 s 置为均匀分布于(0,1)间的随机矩阵
  a=a.rand()*3-1.5;                          //将 a 置为均匀分布于(-1.5,1.5)间的随机矩阵
  vs=ginv(s);                                //计算 s 的广义逆 s+,存放于 vs 中
  vvs=ginv(vs);                              //计算 vs 的广义逆 vs+=(s+)+,存放于 vvs 中
  d=eig(a,v);                                //计算 a 的特征值 d 与特征向量 v
  av=Complex(a)*v;                           //计算 a*v,存放于 av 中
  vd=v*d;                                    //计算 v*d,存放于 vd 中
  cout<<"matrix s:\n"<<s<<"\n";              //以下各行输出计算结果....
  cout<<"matrix vs=s+:\n"<<vs<<"\n";
  cout<<"matrix vvs=(s+)+:\n"<<vvs<<"\n";
  cout<<"matrix a:\n"<<a<<"\n";
  cout<<"matrix d:\n"<<d<<"\n";
  cout<<"matrix v:\n"<<v<<"\n";
  cout<<"matrix av:\n"<<av<<"\n";
  cout<<"matrix vd:\n"<<vd<<"\n";
  return 0;
}
```

下面是取 n1=2、m1=3、n2=3 时,上述程序的一组运行结果:

```
matrix s:
{[0.010559,0.003967,0.335154]
 [0.033265,0.355724,0.2172]}
matrix vs=s+:
{[-0.044974,0.209461]
 [-1.831707,2.815965]
 [3.0068,-0.039933]}
matrix vvs=(s+)+:
{[0.010559,0.003967,0.335154]
 [0.033265,0.355724,0.2172]}
matrix a:
{[0.11092,-0.912671,0.601016]
 [1.349757,-0.675634,-0.167135]
 [-1.173055,0.594699,0.193045]}
matrix d:
{[(-0.202324,1.350836),(0,0),(0,0)]
 [(0,0),(-0.202324,-1.350836),(0,0)]
 [(0,0),(0,0),(0.032978,0)]}
matrix v:
{[(-0.238895,-0.729404),(-0.238895,0.729404),(0.407955,0)]
 [(-0.740761,0.058219),(-0.740761,-0.058219),(0.581333,0)]
```

```
 [(0.639028, -0.068372),(0.639028,0.068372),(0.829876,0)]|
matrix av:
|[(1.033639, -0.175133),(1.033639,0.175133),(0.013453,0)]
 [(0.071229, -1.012426),(0.071229,1.012426),(0.019171,0)]
 [(-0.036931,0.877055),(-0.036931, -0.877055),(0.027367,0)]|
matrix vd:
|[(1.033639, -0.175133),(1.033639,0.175133),(0.013453,0)]
 [(0.071229, -1.012426),(0.071229,1.012426),(0.019171,0)]
 [(-0.036931,0.877055),(-0.036931, -0.877055),(0.027367,0)]|
```

MATCLIB 所具备的各种功能实际上只反映了其特色的一个方面，它的真正优势在于 OOP 开发方法赋予它的良好扩充能力和对不同开发环境的适应能力。每个应用软件开发者由于工作背景的差异可能有完全不同的功能需求，任何通用软件都不可能一一满足，若所提供的软件可由用户自己按需要扩充和裁减，则可在很大程度上弥补这一不足。MATCLIB 已基本实现了这一点。MATCLIB 除了用作辅助科学计算外，还可当作一个软件资源管理系统，将新开发的软件和已有的 C/C++ 语言子例程(通过包装函数)并入 MATCLIB，由类库统一组织和管理，能更有效地发挥各软件的作用，对程序的检索、使用、更新以及维护软件体系的一致性都十分有利。

第二章　MATCLIB 相关技术与辅助类

本章讨论设计 MATCLIB 类库的各种基础设施，包括类模板、全程变量、全程函数和用于 MATCLIB 错误报告的四个辅助类，它们是开发 MATCLIB 类库的基础。

2.1　MATCLIB 与 C++ 类模板

2.1.1　类模板的特点

模板亦称类属或参数化类型，是 C++ 语言中功能最强的新特性之一。早期的 C++ 编译程序并不支持模板，只允许程序员用宏来定义一些类型参数变化的类，适用范围极其有限。迫切的应用需求和语言自身的发展促进了模板机制的诞生。目前，C++ 标准化组织 X3J16 已将模板列入其工作文件中，并相继出现了多种支持模板的编译系统，Borland C++ 3.1 即为其中之一。

作为一种参数多态机制，C++ 模板提供了建立多个相似类的统一模型，允许使用最低限度的重新编码来实现策略转移，从而能更好地支持重用。模板分为函数模板和类模板，函数模板用于描述定义在多种数据类型上的通用算法，类模板则利用模板参量定义一簇类的通用结构。在类模板中，类与模板参量相关联，给定每一组不同的模板参量值，编译器都将自动生成一个相应的新类，避免了源代码的重复，可显著改善某些软件的开发质量。尽管类模板在表示形式和用法上很象一个类，它和类却存在本质区别。作为类的抽象，类模板的实例是相应于特定模板参量的类而非实例对象，这种类一般称为模板类，它与普通类的用法没有多少区别。

类模板是关于一组类的特征抽象，它强调这些类中与具体类型无关的公共特征，而用变元表征与类型相关的特征。这种机制为类库的开发提供了强有力的支持，可以用少量的重复编码实现不同数据类型的通用算法，大大改善类库的体系结构。以矩阵为例，由于数据类型不同，它有下述不同类型：整数型矩阵、单精度实数型矩阵、双精度实数型矩阵和复数型矩阵，这些矩阵的许多操作是非常相似的，如赋值、加、减、乘、除、分解、合成、对角化等，利用类模板将这些公共特性抽象出来，可避免为每种矩阵重复编写类似代码，提高软件开发效率。在 Borland C++ 4.0 中，包容类库已全部用类模板实现。

类模板是 MATCLIB 的基础，本节将结合 MATCLIB 讨论类模板在使用中的相关问题，包括类模板的定义、引用、继承、重载、实例化等。由于 MATCLIB 仅需处理一个通用参数——向量或矩阵的数据类型 Type(Type 可取为 int、float、double 或 complex)，因而书中示例均只有一个模板参量，多个模板参量的处理方法与此相同。

2.1.2　定义类模板

模板参量是类模板区别于普通类的根本所在。每个类模板均带有一个或多个模板参

量,以特定的形式出现于类模板的定义中。MATCLIB 将矩阵类模板命名为 matrix,它以元素数据类型 Type 为模板参量定义了适合于整数型矩阵、单精度实数型矩阵、双精度实数型矩阵和复数型矩阵的通用处理方法,其定义格式如下:

```
template <class Type>
class matrix
{
    protected:
        ......  //data structure of matrix
    public:
        matrix(int xsize, int ysize, Type init = 0);
        matrix(matrix<Type> &m);
        ~matrix();
        matrix & operator = (matrix &m);
        friend matrix copy(matrix &m1);
        friend int operator ! = (matrix &m1, matrix &m2);
        ......
};
```

2.1.3 实现类模板

(1) 构造函数:

下面的构造函数构造一个类型为 Type、xsize 行、ysize 列的矩阵,并将各元素初始化为 init(缺省时 init = 0):

```
template <class Type>
matrix<Type>::matrix(int xsize, int ysize, Type init)
{
    ......  //implementation detail
}
```

以下为拷贝构造函数:

```
template <class Type>
matrix<Type>::matrix(matrix<Type> &m)
{
    ......  //implementation detail
}
```

(2) 析构函数:

当一矩阵超出其作用域时释放它占用的内存:

```
template <class Type>
matrix<Type>::~matrix()
{
    ......  //implementation detail
}
```

(3) 一般成员函数:

```
template <class Type>
matrix<Type> & matrix<Type>::operator = (matrix<Type> &m)
{
    ......  //implementation detail
}
```

(4) 友元函数：

```
template <class Type>
matrix<Type> copy(matrix<Type> &m1)
{
      ......  //implementation detail
}
template <class Type>
int operator ! = (matrix<Type> &m1, matrix<Type> &m2)
{
      ......  //implementation detail
}
```

2.1.4 引用其它模板类

(1) 在类模板定义中引用其它模板类：

下述示例在类模板 matrix 的定义中引用了模板类 vector<Type>：

```
template <class Type>
class matrix
{
  ......
vector<Type> operator * (vector<Type> &v2);
  ......
};
```

(2) 在类模板实现中引用其它模板类：

```
template <class Type>
vector<Type> matrix<Type>::operator * (vector<Type> &v2)
{
      ......  //implementation detail
}
```

2.1.5 类模板的友元

可以申明类模板 A 作为其它类 B 的友元，此时由该类模板 A 产生的任何函数都将成为类 B 的友元，这里的类 B 可为模板类，也可为普通类。下述实例将类模板 matrix 申明为类模板 vector 的友元：

```
template <class Type>
class vector
{
```

```
    ......
    friend class matrix<Type>;
    ......
};
```

2.1.6 类模板的继承

(1) 类模板作为另一类模板的基类:

与普通类的继承情况类似,导出类模板将拥有自身定义和继承自基类的数据与方法。下面为一简单示例,给出了这种情况下导出类的定义和实现框架,其中类模板 gmat 以类模板 matrix 为基类。作为一新的类模板,gmat 同样可适用于不同的数据类型。

```
template <class Type>
class gmat:public matrix<Type>
{
    gmat(int xsize, int ysize, Type init = 0.0):
            matrix<Type>(xsize, ysize, init){};
    gmat(gmat &m):matrix<Type>(m){};
    gmat & operator = (gmat &m);
    ......
};
template <class Type>
gmat<Type> & gmat<Type>::operator = (gmat<Type> &m)
{
    ......  //implementation detail
}
```

(2) 类模板生成的类作为另一普通类的基类:

此时导出类首先以特定的模板参量实例化类模板,得到一模板类,然后对其数据和方法予以继承,因而其效果与普通类的继承一致。在下面的例程中,cmatrix 继承了类模板 matrix 的一个实例(或由它生成的一个模板类)matrix<complex>,经过扩充复数矩阵的专用处理方法,得到用于复数矩阵运算的普通类,它的数据结构和许多基本运算方法均从模板基类中继承。

```
class cmatrix:public matrix<complex>
{
    cmatrix(int xsize, int ysize, complex init = 0.0):
            matrix<complex>(xsize, ysize, init){};
    cmatrix(cmatrix &m):matrix<complex>(m){};
    cmatrix & operator = (vector<complex> &m);
    ......
};
cmatrix & cmatrix::operator = (vector<complex> &m)
{
    ......  //implementation detail
}
```

2.1.7 类模板中的重载机制

类模板中的重载亦分为函数重载和操作符重载。重载的操作符可作为成员函数，也可用作友元函数。下例中操作符“*”作为友元函数被重载，因而它既可用于相容维数的二矩阵相乘，也可用于矩阵与向量相乘。

```
template <class Type>
class matrix
{
      ......

        friend matrix operator * (matrix &m1, matrix &m2);
        friend vector<Type> operator * (matrix &m1, vector<Type> &v2);

      ......
};

template <class Type>
matrix<Type> operator * (matrix<Type> &m1, matrix<Type> &m2)
{
      ......  //implementation detail
}

template <class Type>
vector<Type> operator * (matrix<Type> &m1, vector<Type> &v2)
{
      ......  //implementation detail
}
```

下面是类模板成员函数重载示例，经重载的函数 prod 既可计算二矩阵对应元素相乘，也可用于一矩阵各元素与一常量相乘。由于它是类模板成员函数，因而依模板参量的不同，可用于不同类型的矩阵。

```
template <class Type>
class matrix
{
      ......

      matrix prod(matrix &m2);
      matrix prod(Type x);
      ......
};

template <class Type>
matrix<Type> matrix<Type>::prod(matrix<Type> &m2)
{
      ......  //implementation detail
```