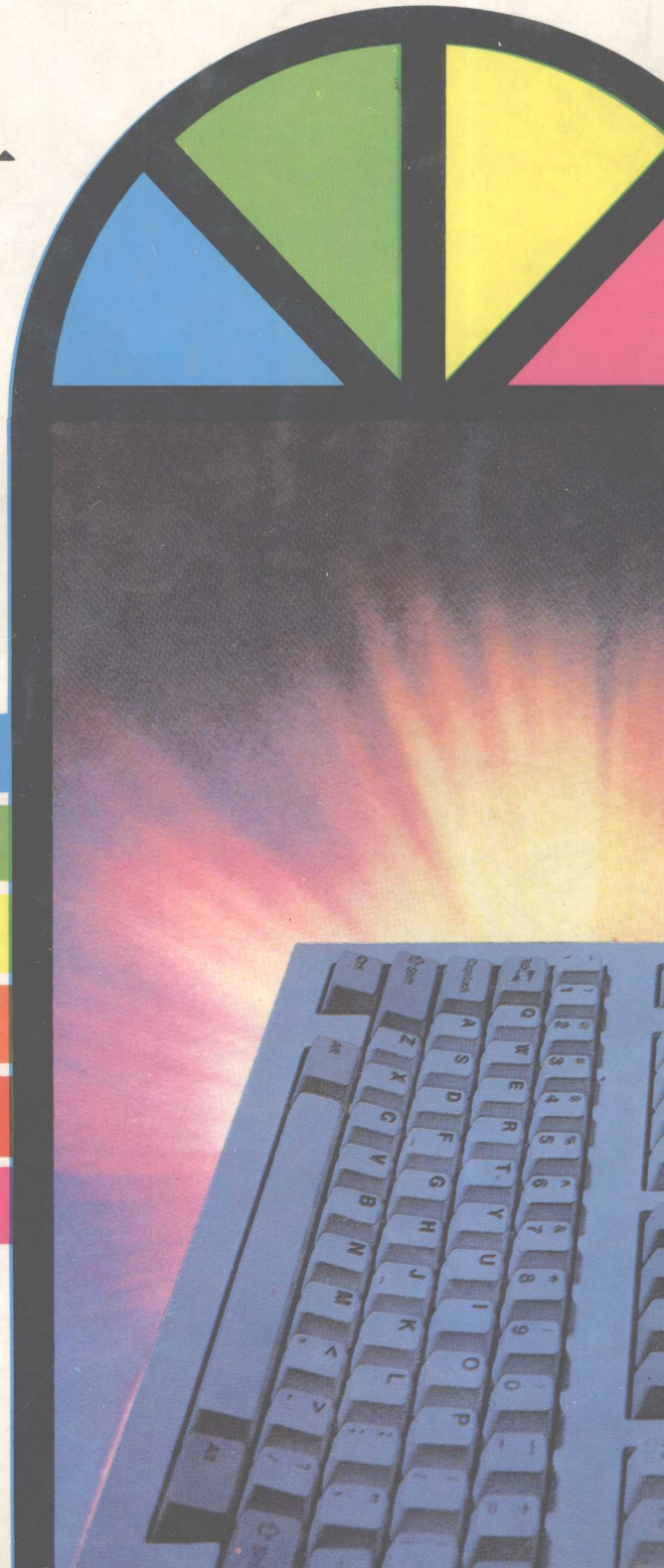
计算机及软件技术丛书

VISUAL C++面向对象的通用类与通用算法设计

魏　宁　张志明　等编著
潘金贵　徐殿祥　审校



南京大学出版社

计算机及软件技术丛书

Visual C++ 面向对象的通用类与通用算法设计

魏　宁　张志明　等编著
潘金贵　徐殿祥　审校

南京大学出版社

1995·南京

（苏）新登字 011 号

内 容 简 介

Visual C++ 是目前流行的高级语言程序设计环境。本书从我们所熟悉的数据结构着手，讨论如何设计和实现自己的通用类。书中涉及到的十种常用的数据结构是：堆栈、队列、数组、矩阵、单向链表、双向链表、图、AVL 树、奇异矩针和杂凑表。在讲述每一种数据结构时，先介绍这种结构的概念，再给出定义这种结构所需的数据成员及其操作（成员函数），进而用 Visual C++ 给出结构的类定义，最后用测试程序对通用类进行测试，并演示它的使用方法。

值得指出的是，书中给出的类都是通用类，而且适用于任何数据类型和任何数据大小。此外，对书中的程序作一些改动，即可使之在其它 C++ 环境下运行。

对 C++（尤其是 Visual C++）语言程序员而言，本书是一本很实用的技术指导书籍。本书可作为数据结构、程序设计方法、通用程序设计技术、C++ 或 Visual C++ 等方面的培训教材。

计算机及软件技术丛书

Visual C++ 面向对象的
通用类与通用算法设计

魏 宁 张志明 等编著
潘金贵 徐殿祥 审校

南京大学出版社出版
（南京大学校内，邮码：210093）
江苏省新华书店发行 江苏省武进县第三印刷厂印刷

开本：787×1092 1/16 印张：19 字数：480 千
1995 年 5 月第 1 版 1995 年 5 月第 1 次印刷
印数：1—5000

ISBN 7-305-02703-0 / TP · 103
定价：20.00 元

《计算机及软件技术丛书》编委会

出版者的话

我国社会主义经济建设的蓬勃发展，极大地推动着社会信息化的进程，也促进了信息产业的发展。现在，计算机的应用已渗透到社会和生活的各个领域。作为社会信息化基础的计算机及软件技术，正为越来越多的人掌握和应用，计算机及软件技术也因此而不断更新、发展。

掌握计算机技术，是现代人特别是跨世纪的中青年人在当今激烈的社会竞争中制胜的基础，也是未来信息化社会对每个人的要求。然而，在我国，计算机基础教育尚欠普及，计算机特别是微型计算机及软件技术的应用和开发也还处在一个较低的层次。许多非专业人员希望能使用计算机，但面对纷繁的专业知识，众多的技术资料，视学习计算机的使用为畏途，专业人员面对软件技术的快速更新，目不暇接。为了让更多的人熟悉计算机技术，利用计算机服务于自己的管理、科研、教学工作，使我国的计算机及软件技术的应用和开发紧随国际潮流，普及和提高我国计算机应用和开发的水平，我们为此组织编写并陆续出版《计算机及软件技术丛书》。

本《丛书》将以应用为基础，兼顾普及与提高。组织科研、教学和应用开发第一线的专家、学者，结合国外计算机及软件技术的最新发展和趋向与国内的应用现状和方向，为初学者提供系统的入门读物，为专业人员介绍适合国情的最新实用技术，既有理论性、学术性强的专著、专论，也有普及性、实用性的教材、手册，以满足多层次读者的需要。

本《丛书》的编写将立足于现实，着眼于未来，力争反映国内外计算机及软件技术的最新动态和发展趋向，引导和帮助读者学习、吸收、掌握计算机的新理论、新技术和新成果。

我们将根据读者需要，不断充实、完善本《丛书》内容，同时诚恳欢迎读者对本《丛书》提出建议、批评，也热忱欢迎向本《丛书》赐稿。

南京大学出版社

《计算机及软件技术丛书》编委会

前　言

Visual C++ 是目前流行的高级语言程序设计环境。相对于其它语言（如 Borland C++ 和 Microsoft C/C++）来说，Visual C++ 除了具有方便的用户界面之外，其中所包含的丰富的类库更是它的一大特色。但是，本书并不讲述如何使用 Visual C++ 类库，而是从我们所熟悉的数据结构着手，讨论如何设计和实现自己的通用类。

学过数据结构或程序设计方法的读者大多都阅读过经典著作《数据结构 + 算法 = 程序》，本书就是以这本书为框架编写而成的。不同的是，本书加进了面向对象和 Visual C++ 的有关内容。书中涉及到的十种常用的数据结构是：堆栈、队列、数组、矩阵、单向链表、双向链表、图、AVL 树、奇异矩阵和杂凑表。在讨论每一种数据结构时，先介绍这种结构的概念，再给出定义这种结构所需的数据成员及其操作（成员函数），进而用 Visual C++ 给出结构的类定义，最后用测试程序对通用类进行测试，并演示它的使用方法。在讨论链表和 AVL 树时，我们还利用自己定义的通用类编制了一个 DOS 文件表实用程序，它可以模拟相关的 DOS 功能。

值得指出的是，书中给出的类都是通用类，而且适用于任何数据类型和任何数据大小。此外，对书中的程序作一些改动，即可使之在其它 C++ 环境下运行。

对 C++（尤其是 Visual C++）语言程序员而言，本书是一本很实用的技术指导书籍。本书可作为数据结构、程序设计方法、通用程序设计技术、C++ 或 Visual C++ 等方面的培训教材。

本书第一到第四章由魏宁、张志明执笔；第五到第七章由刘秀英、何大庆执笔；第八到第十一章由李纪洪、朱尽染、李蕾执笔。全书由潘金贵和徐殿祥老师审校。此外，魏忠才、程功为本书的编排付出了辛勤的劳动。在此对以上同志深表感谢。

限于水平和时间仓促，书中的错误和不妥之处，请读者批评指正。

编　者

1994 年 10 月于南京

目　　录

第一章　什么是通用程序

软件的重用、维护和更新是程序员经常要面临的问题。结构化程序设计为在多个程序中重复使用同一例程开辟了道路。最好是能寻求一种方法，以便不加修改地重复使用代码。当然，代码经常要经过某些修改，以便新的应用程序能够使用它。例如，对某个程序中操作整型数组的例程，可以加以修改，以便用于在另一个程序中操作字符串数组，结果我们最终不得不保留多份非常类似的代码。

上述例子也适用于其他数据结构，如表、栈、队列和树等，程序员经常要借助这些数据结构完成大量的数据处理任务。通常的作法是参照某个现有的数据结构，对其代码进行加工，然后应用到一个新的程序中去。这样，最终得到的一大堆既浪费空间、又难以更新和维护的例程便成了一个负担。

为了解决上述问题，第一步是要开发通用数据结构（如通用数组），并使它具有一些公共的功能，如查找和排序等。通用数据结构的各实例之间互不相同，即其基本数据类型是不同的。此外，数据结构自身固有的参数也可能有所不同。

这样做的好处在于，同一结构的不同实例可以共享相同的代码，因而极大地简化了维护和更新工作，使程序员可以只处理一份代码，而对代码的修改却可以影响所有现有的客户应用程序（在重新编译之后）。因此，通用程序设计所采用的“中心代码”方案能够有效地控制应用程序的开发。Ada 等计算机通常支持通用程序设计，而 C，Pascal 和 Modula－2 则通过其基本语言特征对通用程序设计提供了隐含支持。

通过使用面向对象的程序设计，甚至还可以减少通用数据结构的编码量。对于具有相似数据结构的类（如有序表和无序表），可以通过继承来共享例程和代码，从而实现某些公共功能。例如，清空有序表与清空无序表的过程就是相同的。因此，相关类可以继承某些操作，这样编码量就大大减少了。

在后面我们将讨论动态数组、堆栈、队列、表、杂凑表、树及图等通用数据结构。

1.1　通用程序设计概述

考虑下面的 C＋＋ 函数 SearchInt，它使用线性查找技术来查找整型数组中的某个整数，并返回该整数的下标值；若查找失败，则返回 0xffff。该函数只能处理整型数组。

```
unsigned SearchInt (int A[ ], int Key, unsigned n)
{
    unsigned i = 0;
    int notfound = 1;
    while (notfound && i <= n) {
        if (A[i] == Key)
            notfound = 0;
```

```
        else
            i++;
    }
    return (notfound != 0) ? i : 0xffff;
}
```

可以方便地将此函数改成能够处理长整型数组。下面给出 SearchInt 函数的长整型数组版本，即 SearchLong：

```
unsigned SearchLong (long A[ ], long Key, unsigned n)
{
    unsigned i = 0;
    int notfound = 1;

    while (notfound && i <= n)  {
        if (A[i] == Key)
            notfound = 0;
        else
            i++;
    }
    return (notfound != 0) ? i : 0xffff;
}
```

现有两个相似的线性查找函数：一个处理整数，另一个处理长整数。

下面再考虑一个用以维护 DOS 文件数组的应用程序。这里使用了 find_t 结构，它声明于头文件 DOS.H 中。既可以使用 SearchInt，也可以使用 SearchLong 来扫描 find_t 结构数组。除了修改函数名和参数类型之外，对 if 语句也作了少量修改，以便比较 find_t 结构中的文件名成员。下面给出线性查找函数的另一个版本：

```
#include <string.h>
#include <dos.h>

unsigned SearchDosFileName (_find_t A[ ], char * Key, unsigned n)
{
    unsigned i = 0;
    int notfound = 1;

    while (notfound && i <= n)  {
        if (strcmp(A[i].name, Key) == 0)
            notfound = 0;
        else
            i++;
    }
    return (notfound != 0) ? i : 0xffff;
}
```

依此类推，可以创建许多版本的线性查找函数，它们用来处理一种预定义的或用户定义的数组类型或一种维数各不相同的数组类型。可以创建无穷多个这样的函数，但是，怎样维护它们呢？太让人望而生畏了！解决方法是什么？这正是本书所要回答的。

1.2 建立通用例程

维护多个版本的相似例程是一项既费时、又费事的工作。其实，上述三个版本的线性查找函数可由一个通用函数代替，但要求这个通用函数能够处理不同维数的数组和不同的基本数据类型。

要建立能够处理数组或其他数据结构的通用例程，可以遵从下列方法：

(1) 设置一个指向数组或其他数据结构的基地址的指针。该指针要么是通用的 void * 类型，要么是用户定义的指针类型，这依赖于数据存取方案。一些数据结构既适用于用指针来存取，也适用于用用户定义的指针类型来存取。

(2) 考虑基本元素的大小，这是很有必要的，因为设计通用例程的目的就是为了处理不同的数据大小。

(3) 考虑结构的大小，这依赖于数据结构的类型。数组、矩阵及图等结构就需这类信息。另一方面，如堆栈、队列及表等结构则不需要这类信息。

(4) 指派通用例程中的数据，这是由 memmove 函数完成的，其通用语法如下：

```
memove (destinationPointer,sourcePointer,elementSize);
```

1.2.1 通用线性查找函数

下面的通用函数可以对任意类型、任意维数的数组进行线性查找：

```
unsigned GenSearch (void * A, void * Key, unsigned ElemSize,
                    unsigned n, int ( * CmpFunc)(void * , void * ))
{
    unsigned i = 0;
    int notfound = 1;
    void * p = A;

    while (notfound && i <= n)  {
       if (( * CmpFunc)(p, Key) == 0)
          notfound = 0;
       else
          p+ = ElemSize;
    }
    return (notfound ! = 0) ? i : 0xffff;
}
```

现在，先剖析该函数的参数表：

```
unsigned GenSearch (void * A, void * Key, unsigned ElemSize,
                    unsigned n, int ( * CmpFunc)(void * , void * ))
```

参数 A 是指向客户数组基地址的指针，而不是一个数组类型。参数 Key 是指向所要查找的关键字的指针。这两个指针都是 void * 类型。与参数 A 及 Key 相关的类型确保了它们不会指向任何特定的数据类型。参数 ElemSize 指定每个数组元素的字节数，它也表示关键字

的大小。该函数保留了非通用查找函数中的参数 n。上述函数的功能是比较两个数据项，并返回下列结果：

(1) −1：第一个数据项小于第二个数据项。

(2) 0：第一个数据项等于第二个数据项。

(3) +1：第一个数据项大于第二个数据项。

比较函数的参数必须都是 void 指针，且每个比较函数都必须把该指针强制转换成所要比较的实际类型，以便对由指针形参传递的实参进行比较。

现在，再回过头来看看通用线性查找函数，分析一下它是如何存取数组元素的。参数 A 的地址被赋给局部指针 p，这个操作不是多余的，而是一种很好的程序设计习惯，它相当于执行拷贝操作。开始时，指针 p 中存放的是第一个数组元素的地址。该例程使用 (* CmpFunc)(p,Key) 来比较数组元素与关键字，其中 CmpFunc 的两个参数均为指针。else 语句负责存取下一个数组元素，该语句递增偏移量值，使其加上每个元素的大小 ElemSize，这一操作可以让 p 指向下一个数组元素。

1.2.2 通用库单元

程序清单 1−1 给出了头文件 GENERIC.H 中的声明，程序清单 1−2 给出了通用库文件 GENERIC.CPP 的源代码，其中含有一系列比较函数（分别对应于各种数据类型），也含有本书后面将会用到的许多杂项函数。该库单元既包含简单类型的比较函数，也包含结构类型（如 tm 及 find_t ）的比较函数。

```
//  程序清单 1−1  头文件 GENERIC.H
/*==========================================
目的：提供一些基本通用例程的原型声明
==========================================*/

#ifndef GENERIC_HPP
#define GENERIC_HPP

/***** 比较函数 *****/
/***** 预定义的简单类型 *****/
int comparestr(const void * d1, const void * d2);
int comparedouble(const void * d1, const void * d2);
int compareint(const void * d1, const void * d2);
int compareword(const void * d1, const void * d2);
int comparelong(const void * d1, const void * d2);
int comparebyte(const void * d1, const void * d2);

/***** DOS 库单元所输出的一些结构 *****/
int comparedates(const void * d1, const void * d2);
int comparetimes(const void * d1, const void * d2);
int comparedatetimes(const void * d1, const void * d2);
int comparedosfilenames(const void * d1, const void * d2);
int comparedosfilesizes(const void * d1, const void * d2);
int comparedosfiletimes(const void * d1, const void * d2);
int comparedosfiledates(const void * d1, const void * d2);
int comparedosfiledatetimes(const void * d1, const void * d2);
```

```
/* * * * * 杂凑函数的原型 * * * * */
unsigned strhashfunc0(const void * d, unsigned hashentries);
unsigned strhashfunc1(const void * d, unsigned hashentries);

#endif

#define GENERIC_HPP

//  程序清单 1-2  库文件 GENERIC.CPP
/*==========================================
目的:提供一些通用库单元的定义
==========================================*/

#include "comndata.h"
#include <string.h>
#include <math.h>
#include <dos.h>
#include <time.h>

int retVal(int i)
{
    if (i > 0)
        return 1;
    else if (i < 0)
        return -1;
    else
        return 0;
}

//------ 字符串的比较 ------

int comparestr(const void * d1,const void * d2)
{
    char * s1 = (char * ) d1;
    char * s2 = (char * ) d2;
    int i = strcmp(s1,s2);

    return retVal(i);
}

//------- 双精度浮点数的比较 -------

int comparedouble (const void * d1,const void * d2)
{
    if ( * (double * ) d1 > * (double * ) d2)
        return +1;
    if ( * (double * ) d1 < * (double * ) d2)
        return -1;
    else
        return 0;
}

//------ 整数的比较 ------

int compareint(const void * d1,const void * d2)
```

```
{
    int i = ( * (int * ) d1 - * (int * ) d2);
    return retVal(i);
}

//------ 字的比较 ----

int compareword(const void * d1,const void * d2)
{
    int i = ( * (unsigned * ) d1 - * (unsigned * ) d2);
    return retVal(i);
}

//------ 长整数的比较 ------

int comparelong(const void * d1,const void * d2)
{
    long i = ( * (long * ) d1 - * (long * ) d2);
    return retVal(int(i));
}

//------ 字节的比较 ------

int comparebyte(const void * d1,const void * d2)
{
    int i = ( * (byte * ) d1 - * (byte * ) d2);
    return retVal(i);
}

//------ 杂凑函数之一 ------

unsigned strhashfunc0(const void * d,unsigned hashentries)
{
    /* 杂凑函数所用的常量:素数 */
    const unsigned hash_const1 = 13;

    unsigned i,most,sum,shift;
    char * p;

    p = (char * ) d;
    shift = 'A' - 1;
    most = strlen(p);
    most = (most > 3) ? 3 :most;
    i = 1;
    sum = 0;

    while ((i <= most))
        sum = sum * 7 + * (p + i++) - shift;

    // 使用了两个 % 操作符,使得杂凑地址不会超出范围
    return (sum % hash_const1) % hashentries;
}

//------ 杂凑函数之二 ------

unsigned strhashfunc1(const void * d,unsigned hashentries)
```

```
{
    /* 杂凑函数所用的常量:素数 */
    const unsigned hash_const1 = 11;
    const unsigned hash_const2 = 17;

    double sum;
    unsigned i,shift,len,k1;
    char * p;

    p = (char * )d;
    shift = 'A' - 1;
    sum = 0.0;
    len = strlen(p);
    k1 = hash_const1 * (len * len) % hash_const2;

    for(i = 0;i < len;i++)
        sum = sum * k1 + sqrt(double( * (p + i)) - shift);

    i = unsigned (sum) % hashentries;

    i = (i % hash_const1) * hash_const2 + (i % hash_const2);

    return i % hashentries;        // 使杂凑地址不会超出范围
}

//------ 获取日期和时间 ------

void get_datetime(_find_t f,tm& t)

/*-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

本例程的目的:分析一个文件的日期和时间

参数:
   输入:f - 文件块数据
   输出:t - 已分析出的某个文件的日期和时间

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-*/

{
    t.tm_sec = f.wr_time & 0xf;
    t.tm_min = f.wr_time & 0x3f0;
    t.tm_hour = f.wr_time >> 11;
    t.tm_mday = f.wr_date & 0x1f;
    t.tm_mon = f.wr_date & 0x1e0;
    t.tm_year = 1980 + f.wr_date >> 9;
}

//------ 对日期进行比较 ------

int comparedates(const void * d1,const void * d2)

/*-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

本例程的目的:比较两个日期数据
```

```
参数:
    输入:d1,d2 - 指向日期时间类型变量的指针

返回值:
    datetime(d1) 的日期 > datetime(d2) 的日期: 1
    datetime(d1) 的日期 < datetime(d2) 的日期: -1
    datetime(d1) 的日期 == datetime(d2) 的日期: 0

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-*/

{
    tm datetime1,datetime2;

    datetime1 = *(tm*) d1;
    datetime2 = *(tm*) d2;

    if (datetime1.tm_year > datetime2.tm_year)
       return 1;
    else if (datetime1.tm_year < datetime2.tm_year)
       return -1;
    else if (datetime1.tm_mon > datetime2.tm_mon)
       return 1;
    else if (datetime1.tm_mon < datetime2.tm_mon)
       return -1;
    else if (datetime1.tm_mday > datetime2.tm_mday)
       return 1;
    else if (datetime1.tm_mday < datetime2.tm_mday)
       return -1;
    else
       return 0;
}

//------ 比较时间 ------

int comparetimes(const void *d1,const void* d2)

/*-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

本例程的目的:比较两个 datetime 的时间域中的日期和时间

参数:
    输入:d1,d2 - 指向日期时间类型变量的指针

返回值:
    datetime(d1) 的时间 > datetime(d2) 的时间: 1
    datetime(d1) 的时间 < datetime(d2) 的时间: -1
    datetime(d1) 的时间 == datetime(d2) 的时间: 0

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-*/

{
    tm datetime1,datetime2;

    datetime1 = *(tm*) d1;
    datetime2 = *(tm*) d2;
```

```
    if (datetime1.tm_hour > datetime2.tm_hour)
        return -1;
    else if (datetime1.tm_hour < datetime2.tm_hour)
        return -1;
    else if (datetime1.tm_min > datetime2.tm_min)
        return -1;
    else if (datetime1.tm_min < datetime2.tm_min)
        return -1;
    else if (datetime1.tm_sec > datetime2.tm_sec)
        return -1;
    else if (datetime1.tm_sec < datetime2.tm_sec)
        return -1;
    else
        return 0;
}

//—————— 比较日期和时间 —+—+—+—+—+—+—+—+—+—+—+—+—

int comparedatetimes(const void *d1,const void *d2)

/*+—+—+—+—+—+—+—+—+—+—+—+—+—+—+—+—+—
本例程的目的:比较日期时间数据的时间域中的日期和时间

参数:
    输入:d1,d2 — 指向日期时间类型变量的指针

返回值:
    datetime(d1)的日期和时间 > datetime(d2)的日期和时间 : 1
    datetime(d1)的日期和时间 < datetime(d2)的日期和时间 : -1
    datetime(d1)的日期和时间 == datetime(d2)的日期和时间 : 0

+—+—+—+—+—+—+—+—+—+—+—+—+—+—+—+—+—*/

{
    tm datetime1,datetime2;

    datetime1 = *(tm *) d1;
    datetime2 = *(tm *) d2;

    if (datetime1.tm_year > datetime2.tm_year)
        return 1;
    else if (datetime1.tm_year < datetime2.tm_year)
        return -1;
    else if (datetime1.tm_mon > datetime2.tm_mon)
        return 1;
    else if (datetime1.tm_mon < datetime2.tm_mon)
        return -1;
    else if (datetime1.tm_mday > datetime2.tm_mday)
        return 1;
    else if (datetime1.tm_mday < datetime2.tm_mday)
        return -1;
    else if (datetime1.tm_hour > datetime2.tm_hour)
        return 1;
    else if (datetime1.tm_hour < datetime2.tm_hour)
        return -1;
```