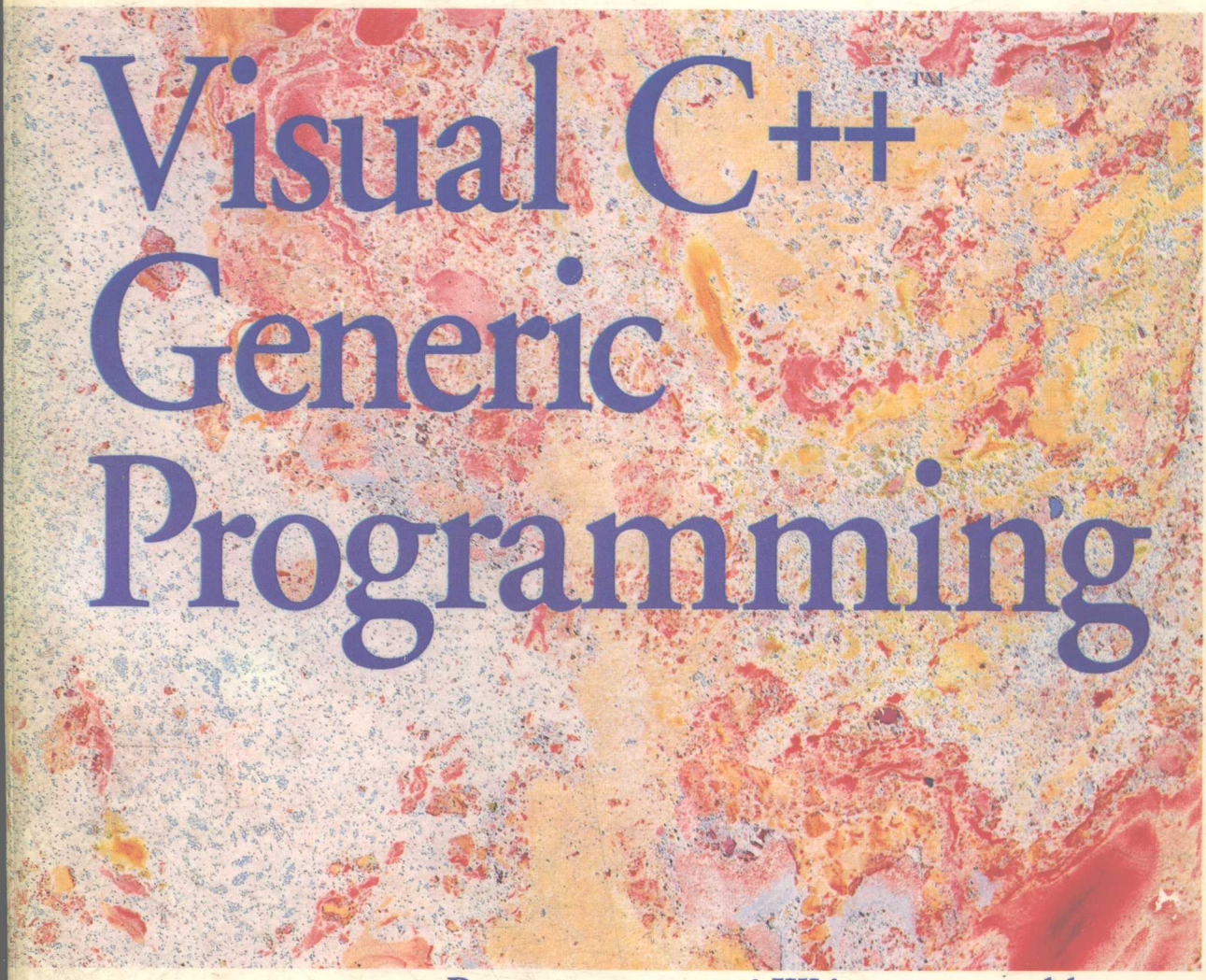
计算机语言技术系列丛书(二)

Visual C++类属编程实例



- Program smarter! Write more and better code in less time
- Compatible with QuickC and C++ 7.0
- Includes all source code and project files on disk

Namir C. Shammas 著

张 勇 译

学苑出版社

计算机语言技术系列丛书(二)

Visual C++ Generic Programming

Visual C++类属编程实例

[美] Namir C. Shammas 著
张 勇 译
燕卫华 审校

学苑出版社
1994.

(京)新登字 151 号

内 容 提 要

本书主要介绍开发应用程序和工具常要涉及到各种数据结构，如数组、表和栈。该书为正在学习C++的程序员提供了具有类属数据结构模型的类库，该类库使程序员只保留单一版本的栈、队、数组、矩阵、链表、二叉树、散列表和图。

本书具有很强的实用性，书中附有大量的范例，并配有包含这些范例的软盘。需要本书软盘的读者可直接直与北京 8721 信箱资料部联系，邮码 100080，电话 2562329。

软盘单价：50.00 元(含邮费)。

计算机语言技术系列丛书(二)

Visual C++ Generic Programming

Visual C++类属编程实例

原　　著：Namir C. Shammas
翻　　译：张　勇
审　　校：燕卫华
责任编辑：甄国宪
出版发行：学苑出版社
邮政编码：100036
社　　址：北京市海淀区万寿路西街 11 号
印　　刷：北京市地质矿产局印刷厂印刷
开　　本：787×1092　1/16
印　　张：23　　**字 数**：545 千字
印　　数：1～5000 册
版　　次：1994 年 8 月北京第 1 版第 1 次
ISBN7－5077－0905－1/TP・29
本册定价：48.00 元

用 户 请 注 意

欲购本书配套软盘的朋友，请按下列方法汇款：

单价：50.00 **元（含邮费）**

注：从银行电汇款的朋友请按下列帐号和收款单位汇款：

收 款 单 位：北京希望电脑公司
开户银行及帐号：北京海淀工商行中关村城市信用社　　帐号：05079－08
注：要增值税发票的朋友，请仔细填下表。

购 货 单 位	名称	纳税人登记号
	地址电话	开户银行及帐号

注：本次共订盘　　张　　应收款为￥（小写）：

注：用户填好此单后请连同此单、信汇单一并传真 01－2561057 或 01－2579874，收到传真后即发货。或邮寄 100080 北京海淀 8721 信箱资料部朱红收

联系电话：01－2562329，2541992

序　言

开发应用程序和工具常要涉及到各种数据结构，如数组、表和栈。本书为正在学习的C＋＋程序员提供了具有类属数据结构模型的类库，该类库使程序员只保留单一版本的栈、队、数组、矩阵、链表、二叉树、散列表和图。

本书共分十一章，不仅阐述了编写类属类的基础知识，而且介绍上面提到的通用数据结构类。

第一章介绍开发类属数据结构类采用的基本方法，及创建类属数据结构的C语言风格方法。

第二章着眼于类属栈——其基本操作和设计需求。本章介绍抽象的、基于堆的栈的类属类。

第三章研究类属队——其基本操作和设计需求。本章讨论队的基本类型，并介绍一种具有灵活队模型的类，可以由两端生长，形成栈。

第四章研究类属数组。本章讨论数组的基本操作以及类属数组的设计说明。本书介绍了一个抽象类和一个基于堆的类属数组类，并且使用了几个例子说明如何将类属数组类同数、串及用户定义的类一起使用。

第五章介绍类属矩阵并讨论其基本操作和设计参数。本章介绍一个抽象类属矩阵类及一个基于堆的类属矩阵类。

第六章讨论通用的类属散列表，并探讨速度优势、基本操作及使用该表需要的结构。此外，本章引入一个类属散列表类，用它可以快速建立一个交叉引用名表。

第七、八章分别讨论单向链表和双向链表。这两章很类似，均详细地研究了链表的基本操作和设计需求。第七章还介绍了一个能建立DOS文件列表的特殊类，该类常被DOS工具使用，该章还能介绍了一个显示文件的名、大小和日期/时间的程序，且可以匹配多个通匹符。

第九章介绍了通用的准平衡AVL树。AVL树是二叉树的超类型，可以防止树的倾斜。本章讨论AVL树的基本操作、设计需求，并介绍了具有抽象的和基于堆的AVL树模型的类。本章还介绍了维护DOS文件列表的AVL文件类。

第十章讨论通用类属图。本章着眼于各种图及表示它们的方法。本章还介绍了类属图类，并提供了特殊的例子，包括一个用图解决电子电路的例子。

第十一章讨论特殊的锯齿矩阵。本章着眼于矩阵的结构、基本操作和设计需求。本章还介绍了一个基于堆的类属矩阵类。

译者的话：

在翻译过程中，在原书中发现了一些错误，并且已在译文中改正了。晏海华、杨峰、谢小兵、孙波等参加了本书的翻译工作，在此表示感谢。

目 录

第一章　类属编程概述

软件的可再用性、可维护性和可更新性都是程序员面临的问题。结构化程序设计鼓励一些程序在多个程序中重用。理想的重用模式是不做任何修改就重用代码，然而，为了能在新的应用程序中正常运行，这样的代码通常要进行修改和剪裁。例如，处理整数数组的程序要编辑修改后才能在另一程序中用于处理串数组，结果就有了多段非常相似的代码。

这个例子同样存在于其它数据结构，如表、队、栈和树，程序员就是使用这样多种多样的数据结构来完成各种数据处理任务。程序常要得到某一数据结构的一个已有实例，并且剪裁它的代码以适全新的应用程序，结果是产生了难于更新和维护并耗费磁盘空间的程序。

解决这个问题的第一步是采用类属数据结构，这些数据结构，如类属数组，拥有通常的功能性，如排序和查找。类属数据结构的各种实例因其基本数据类型而彼此不同。此外，它们还常因适合结构本身的其它参数而不同。

类属编程的优点在子同一结构的不同实例共用同样的代码，这就大大简化了维护和更新，因为程序员只需处理单一版本的代码，所做的修改可不重编译就适合所有的已有客户程序，因此集中代码模式在控制软件开发方面很有效。有些语言，如 Ada，支持类属编程。相对而言，其它一些语言，如 C、Pascal 和 Modula－2 通过基本语言特性提供了隐含的支持。

使用面向对象语言程序员在用类属数据结构编码时可减少工作量，相似数据结构的类（如有序表和无序表）使用继承来共享实现公用功能的程序代码，例如，清有序表和无序表就使用同样的过程。随后，相关的类就继承这样的操作，进一步减少代码量。

本书将合理的结构化技术、类属编程以及为通用数据结构实现类属类的 OOP 三方面的优点结合起来。结构包括动态数组、栈、队、表、散列表、树和图，这些结构的类提供了功能很强的省时的库，对于这些通用数据结构可以非常有效地减少代码量。采用 OOP 的类提供了同软件工程最新发展保持一致的有吸引力的格式。

1.1　类属编程元素

请看下面的 C＋＋函数，SearchInt，它采用了线性查找算法在一个整型数组中定位一个整数，该函数返回相应整数的下标，若无匹配的整数则返回 0xffff。该函数只能处理整数。

```
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;
}
```

你可以很容易地使该程序中的代码适合处理长整型数组，只要简单地编辑修改一下函数 SearchInt，产生下面的函数 SearchLong 即可。现在就有了线性查找函数的两个非常相似的版本：一个适合于整型，另一个适合于长整型。

```
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;
}
```

让我们来讨论一下维护 DOS 文件数组的应用程序。程序中使用了结构_find_t，它是在头文件 DOS. H 声明的。可以用函数 SearchInt 或 SearchLong 为结构_find_t 产生另一版本，除了要修改各种参数数据类型外，if 语句还需做少量修改，这些修改适合结构_find_t 的 name 字段的比较。下面的程序清单列出了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 (stricmp(A[i].name, Key) == 0)
    notfound = 0;
  else
```

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

可以这样继续下去，能产生线性查找函数的更多的版本，每个版本分别适合不同的预定义或用户定义类型以及不同大小的数组类型。这样能够产生天文数字个数的版本，然后维护起来就会非常头疼。有什么出路吗？请继续阅读。

1.2　建立类属程序

维护相似程序的多重版本既耗资又易出错。线性查找函数的上面三个版本可由一个类属版本来替代，它可以适合不同大小的数组和基本数据类型。

建立能处理数组和其它数据结构的类属程序的基本组成部分如下：

- 一个指向数组或其它数据结构的基地址。指针类型或者是通用的 void * 类型，或者是用户定义的指针类型，采用哪种取决于数据访问模式。一些数据结构或者处理指针。或者处理用户定义的指针类型。
- 基本元素大小。这是一条重要的信息，因为是在编写处理可变数据大小的类属程序。
- 比较函数，类属程序需要用于比较数据元素。
- 结构的大小，取决于数据结构的类型。像数组、矩阵和图这样的结构需要这一信息。相对而言，其它数据结构，如栈、队、表和树不需要这类信息。
- 类属程序中的数据赋值，这是由函数 memmove 来完成的，其一般的语法为：

memmove (destinationPointer, sourcePointer, elementSize)

1.3　类属线性查找函数

下面的程序给出了一个类属函数，可以完成对任何类型和大小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;
}
```

我们先研究一下参数表：

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

参数 A 是客户数组而不是一个数组类型的基地址，参数 Key 是在查找时指向要用的键数据的指针，也声明为 void * 。与参数 A 和 Key 相关的类型保证了它们不会链接到任何特殊的数据类型上 。参数 ElemSize 表示每个数组元素所占的字节数，它也是键数据的大小。函数保留了查找函数的非类属版本中的参数 n，CmpFunc 是一个指向比较函数的指针，现在我们就着重看看该函数，该函数比较两个数据项并返回如下结果：

- 当第一项小于第二项时，返回－1。
- 当两项相等时，返回 0。
- 当第一项大于第二项时返回＋1。

比较函数的参数必须是两个 void 指针，每个比较函数必须再将这些指针参数强制转换成实际要比较的类型。比较通过指针参数存取的参数时需要这种转换。

再回到类属线性查找函数，看看程序如何存取数组元素。将参数 A 的地址赋给局部指针 p，并非必须如此，但处理一个副本是一个很好的方法。最初，指针 p 具有第一个数组元素的地址，程序用(* CmpFunc)(p,Key)比较那个元素和键数据，CmpFunc 的两个参数均为指针。else 子句存取下一个数组元素，这条语句增加了 ElemSize 大小的偏移量，即每个元素的大小，这一运算使指针 p 存取下一数组元素。

1.4 类属库单元

程序 1-1 给出了头文件 GENERIC. H 中的声明，程序 1-2 给出了类属库文件 GENERIC. CPP 的源码，里面有各种数据类型的全部比较函数，还有几个本书后面要用到的散列函数。库单元中有一些简单类型和结构，如 tm 和_find_t，的比较函数。

1.5 OOP 和类属编程

前面的关于类属编程的讨论遵循了一种结构化编程方法——产生处理数据的类属程序。面向对象程序设计将重点由过程移到了对象上，这意味着，例如，侧重面由处理数组的类属线性函数转到了具有类属线性查找功能的类属数组上，现在主要关心的是类属数组，线性查找已成为第二位(尽管仍很重要)的了。研究类属数组对象的功能性，会发现可以包含其它操作，包括排序、数据存储和数据检查。

程序 1-1　头文件 GENERIC. HPP 的源码：

```
/* ========================================================

        Copyright (c) 1989—1993 Namir Clement Shammas

    Version: 1.1.0                    Date 2/5/93

    PURPOSE: provides a basic repertoire for basic generic routines.

    UPDATE HISTORY:

======================================================== */
#ifndef _GENERIC_HPP
#define _GENERIC_HPP

/*  * * * * * * * * comparison functions * * * * * * * * * * * */
/*  * * * * * predefined simple types * * * * * * * * * * *  */
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);
/* * * * * records exported by dos library unit * * * * * */
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);

/* * * * * * * * * * * hash functions * * * * * * * * * * * * * */
unsigned strhashfunc0(const void * d, unsigned hashentries);
unsigned strhashfunc1(const void * d, unsigned hashentries);

#endif
```

程序 1-2　库文件 GENERIC. CPP 的源码：

```
/*==========================================

        Copyright (c) 1989－1993 Namir Clement Shammas

    Version: 1.1.0                    Date 2/5/93

    PURPOSE: provides a basic repertoire for generic library units.

    UPDATE HISTORY:

==========================================*/

#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;
}

//------------------------ comparestr ------

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

    return retVal(i);
```

```
}

//————————————————————— comparedouble ———————

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;
}

//————————————————————— compareint ———————

int compareint(const void * d1, const void * d2)
{
    int i = ( * (int * ) d1 - * (int * ) d2);

    return retVal(i);
}

//————————————————————— compareword ———————

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

    return retVal(i);
}

//————————————————————— comparelong ———————

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

```
    return retVal(int(i));
}

//---------------------------- comparebyte -------

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

    return retVal(i);
}

//---------------------------- hash0 -------

unsigned strhashfunc0(const void * d, unsigned hashentries)
{
    /* hash function constants: prime numbers */
    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;

    // use two %ulo operators to keep hash address in range
    return (sum % hash_const1) % hashentries;
}

//---------------------------- hash1 ---

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

```
{
    /* hash function constants: prime numbers */
    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; // keep hash address in range

}

//---------- get_datetime -----

void get_datetime(_find_t f, tm& t)

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

ROUTINE PURPOSE: parse the time and date of a file.

PARAMETERS:

  INPUT: f - the file block data.

  OUTPUT: t - the parsed time and date of the file.
```

```
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-*/
{
    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;
}

//--------------------- comparedates -----

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

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

ROUTINE PURPOSE: compares the dates.

PARAMETERS:

  INPUT: d1, d2 - the pointers to the datetime-typed variables.

int value:

    date of datetime(d1) > date of datetime(d2) :  1
    date of datetime(d1) < date of datetime(d2) : -1
    date of datetime(d1) == date of datetime(d2) :  0

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

{
    tm datetime1, datetime2;

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

    if (datetime1.tm_year > datetime2.tm_year)
```