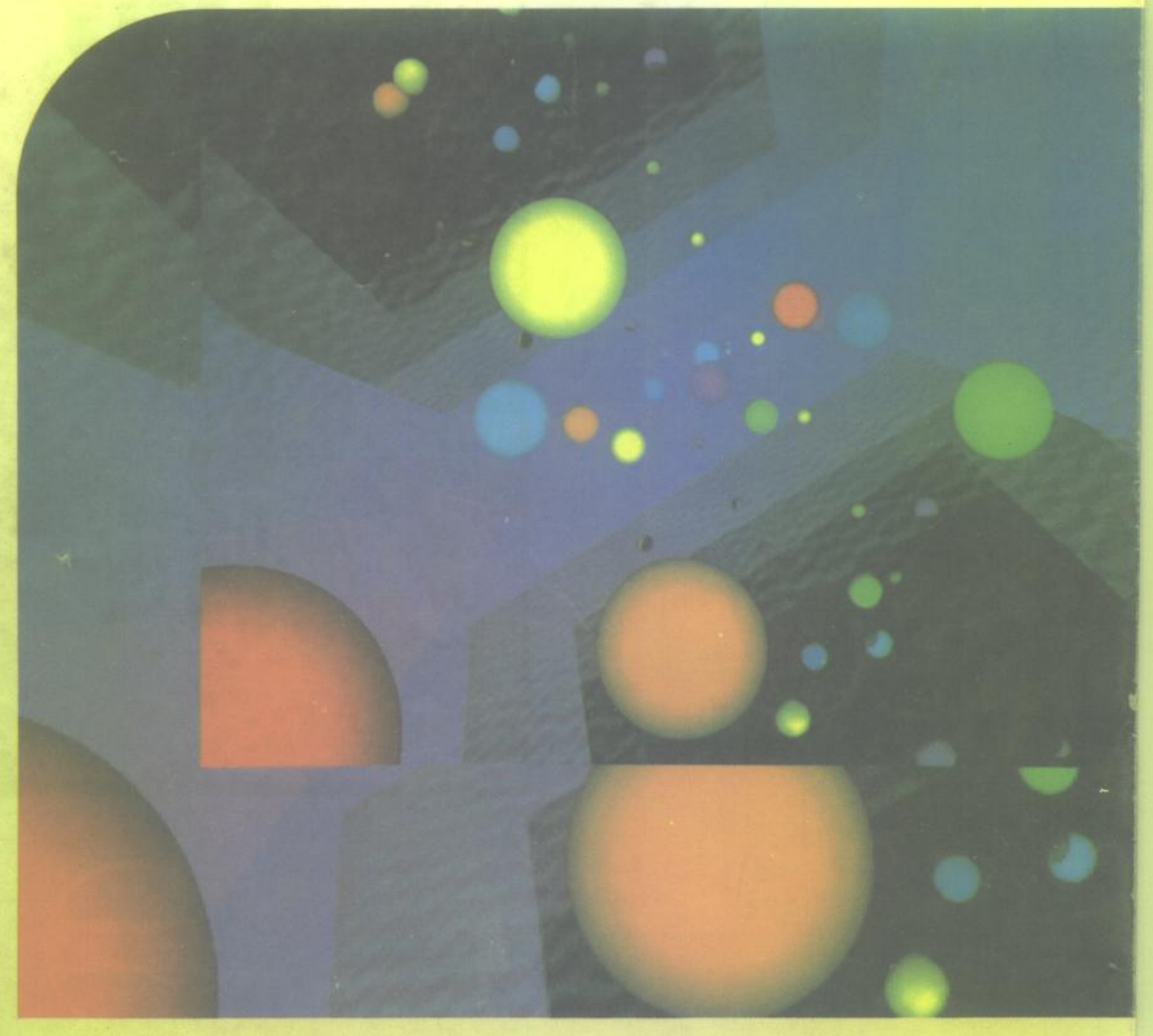
计算机软件应用系列

Visual C++ 程序开发指南（二）

——技术与方法

● 徐 力 等 编著



科学出版社

计算机软件应用系列

Visual C++程序开发指南(二)
—— 技术与方法

徐 力 等 编著

科 学 出 版 社

1995

（京）新登字 092 号

内 容 简 介

本书介绍了采用标准 I/O 的 Visual C++类，用类建立对象、在对象上建立操作等技术；给出了在 Visual C++中使用继承技术，建立类库等的方法；讨论了虚拟函数、多态和 Visual C++模板、异常处理等新技术。最后介绍了用 Microsoft 基础类进行窗口程序设计的技术和方法。

计算机软件应用系列

Visual C++程序开发指南(二)

——技术与方法

徐 力 等编著

责任编辑 刘晓融 留 霞

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

化学工业出版社印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1995 年 6 月第 一 版 开本：787×1092 1/16

1995 年 6 月第一次印刷 印张：29

印数：1—3 150 字数：671 000

ISBN 7-03-004935-7/TP·485

定价：37.70 元

前　言

面向对象程序设计(OOP)是目前流行的程序语言概念,Visual C++是汇集 Microsoft 公司技术精华的主流产品。《Visual C++程序开发指南》是《计算机软件应用系列》丛书中的一套,共四册。这套书从各个方面说明了应如何在 Visual C++环境下以 OOP 的概念设计 DOS 和 Windows 应用程序。

作为一个成熟的软件工程师,面对铺天盖地的 Word,Excel,PowerPoint,FoxPro,Visual Basic 等应用软件,不熟悉 Visual C++是很不明智的,因为借助 Visual C++,软件工程师就可以开发出很多动态链接库,从而为广大用户提供全方位的支持,其商业前景更是无可估量的。

Visual C++是一个面向对象、界面友好的 DOS 和 Windows 程序开发环境。在这个集成环境中开发应用程序时,不一定要手工设计用户界面,而只需选取菜单命令,Visual C++系统就会生成一个可实际运行的 Windows 应用程序框架。此后,程序员就可利用基于 Windows 的 C++编辑器,借助 AppWizard 建立面向对象的应用程序。

除了 AppWizard 外,Visual C++还包含 C++应用程序生成器、基于 Windows 的编辑器、基于 Windows 的面向图形的编程工具、使 C++代码和 Windows 消息及类成员函数相联系的交互式工具、可用来编写 Visual C++文本程序的 QuickWin 库以及预编译的头文件和源文件。当然,类库丰富更是 Visual C++的最大特色。

尽管目前市面上已有一些 Visual C++方面的书籍,但学习 Visual C++的人常常会有下列困惑:

(1) 对 Visual C++缺乏一致的认识,遇到问题总是缺这少那,缺乏一个值得信赖的良师益友。

(2) 只了解 OOP 的直观语法,而对于 OOP 有何好处、为什么 OOP 可取代结构化语言并成为未来程序设计风格的主流,都不太明白。

(3) 无法理解 OOP 对未来程序的管理与维护究竟有何重要性。

(4) 在 Visual C++环境下设计 OOP 程序时,对 Visual C++提供的资源不太清楚。

(5) 不知道如何用 Visual C++和 Windows 解决实际问题。

针对这些问题,这套书从各个方面介绍了 Visual C++集成环境、OOP 概念、类库以及基于 DOS 和 Windows 的 C++应用程序开发方法,并提供了丰富的类模板和大量的应用程序实例。

本套书的内容基本属于中等难度,适用于初、中级读者。不熟悉 OOP 技术和 C++语言的读者可参阅第一册的第一至十三章,这几章主要介绍 Visual C++集成环境和 OOP 概念。经常从事应用程序开发的读者大都比较关心 Visual C++类库和通用类的构造方法,可参阅第二册和第三册的第七至九章,其中第二册主要介绍通用类的构造方法,并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,这为进行大程序开发的用户提供了设计开发环境的手段;第三册第七至九章则主要介绍 Visual C++基础类库,讨论如何使用 Visu-

al C++类库。比较熟悉 OOP 概念而对 C++程序设计技术不太精通的读者可参阅第四册，其中包含大量的程序设计实例。

本书是这套书的第二册，参与编写的人员有：徐力、徐幕回、郭子逸、何国辉、胡卫东、李华清、李杨、魏志国、李蕾、李建、刘心海、刘崇福、李纪鸿、刘石华、赵越、陆静宜、陈楚南、梁友国、陈佳海。全书由吴佳教授和陈忠敏研究员审校。此外，朱小宝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，书中的错误和不妥之处，敬请读者批评指正。

目　录

第一章　类与对象

本章将讨论对象的行为以及这些行为对类及类体系设计的影响。读者将了解到下列各种类和行为：

① 抽象类
② 多态类
③ 可扩展结构
④ 规范类
⑤ 禁止类
⑥ 迟后联编和重联编

注意，本章中的 C++ 程序全部作为 QuickWin 程序进行编译。

1.1　抽象概念与抽象类

抽象是一种从实验系统中过滤掉细节的分析工具，它能使我们将注意力集中在系统发生了什么变化而忽略这种变化是如何发生的。在面向对象的分析和设计方法中，我们把抽象作为研究问题的范围和涉及到的各类操作的有效手段。在这一节中，我们将讨论抽象类及其在类体系设计中的作用。读者将学到如下内容：

(1) 声明 (declare) 抽象类的基本原则。
(2) 在类体系中将抽象类作为基类。
(3) 在子类体系中将抽象类作为基类。

在设计某个类体系时，可以用一个抽象类来说明该类体系中的通用操作。抽象类能够说明各派生类实例中发生了什么，而如何完成一个类的操作则由各派生类自己定义。

抽象类分成两类：纯抽象类和部分抽象类。纯抽象类说明了类体系中各派生类所共有的公共 (public)、保护 (protected) 及私有 (private) 成员函数，这些成员函数的定义体中不包含语句，因此纯抽象类完全不具备任何功能。

部分抽象类说明了全部或部分派生类所共有的公共、保护及私有成员函数和数据成员。此外，部分抽象类实现了一些成员函数，而这些成员函数对全部或部分派生类而言也是共有的。

这两种类型的抽象类具有如下共同特征：

(1) 需要将所有的成员函数及数据成员声明成保护的或私有的。此外，需要在参数表的后面加上“=0”，以告诉 C++ 编译器，我们正在声明一个纯抽象成员函数。

(2) 不由抽象类实现的成员函数必须声明成虚拟函数，这种声明能确保派生类也把这些成员函数声明成虚拟函数，并使用相同的参数表。使参数表保持一致的好处在于能够支持多态行为。就这一方面而言，部分抽象类比纯抽象类更好。

1.1.1 抽象类作为基类

在一个类体系中，抽象类经常是该类体系的根类。当类体系比较简单，即由一个单一的继承链组成时，一个抽象类即可满足需要。当类体系有许多分支时，则可以包含多个抽象类。图 1.1 列出了一个包含多个抽象类的类体系，其中 CAbsType1 类是所有体系分支所共有的。每个分支有一个附加的抽象类，它用于指定该分支特有的许多操作。

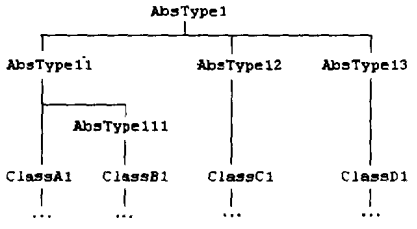


图 1.1 包含多个抽象类的类体系

让我们看一个例子。程序清单 1.1 列出了头文件 ABSSTACK.H，而清单 1.2 则列出了库文件 ABSSTACK.CPP 的源代码，这个库利用一个抽象类实现了一个堆栈类体系。这个库单元支持具有如下基本操作的字符串堆栈：

(1) 清除一个堆栈。

(2) 将数据压入一个堆栈。

(3) 将数据弹出一个堆栈。

这个库声明了一个含有三个类的类体系：CAbsStack，CStrStack 和 CVMStrStack。从名称可以看出，CAbsStack 类是一个抽象类，它将其中所有数据成员及成员函数都声明成保护的。该类是一个部分抽象类，因为它声明了数据成员并包含几个有效的(如果愿意的话，也可以称为“已实现的”)成员函数。数据成员 nHeight 和 bAllocateError 分别存放堆栈的高度和数据分配状态。该类声明了如下成员函数：

(1) 构造函数和析构函数。

(2) 布尔函数 GetAllocateError，它返回 bAllocateError 数据成员的值。

(3) 布尔函数 IsEmpty。当 nHeight 数据成员为 0 时，它返回 TRUE，否则返回 FALSE。

(4) 虚拟函数 Push，它将一个字符串压入堆栈。

(5) 虚拟布尔函数 Pop，它将一个字符串弹出堆栈。

(6) 虚拟函数 Clear，它可以清除堆栈。

在声明后面三个成员函数时使用了一个等号和零标志(=0)，这种语法告诉 C++ 编译器，这些函数是纯虚拟函数。

库文件同样声明 CStrStack 类作为 CAbsStack 的一个派生类，这个派生类模拟一个基于堆的字符串堆栈，实际实现它时使用了一个动态链表。CStrStack 类声明了指针 pTop，以访问(access) 所支持的动态链表。该类声明了一个有效的构造函数、一个虚拟析构函数和虚拟成员函数 Push，Pop 及 Clear，这些成员函数中的语句说明了堆栈操作是如何实现的。构造函数通过初始化所支持的动态链表来初始化堆栈。析构函数用来清除所支持的链表。

库单元也声明了 CVMStrStack 作为 CAbsStack 的另一个派生类，这个派生类模拟一个基

于磁盘的字符串堆栈，实际实现它时使用了一个随机访问的文件流。CVMStrStack 声明了数据成员 szDataBuffer，szErrorMessage 和 VMfile。该类还声明了一个有效的构造函数、一个虚拟析构函数和虚拟成员函数 Push，Pop 及 Clear，这些成员函数中的语句说明了堆栈操作是如何借助所支持的随机访问流文件来实现的。构造函数用于打开所支持的随机访问流文件，而析构函数则用于将成员 nHeight 赋值为 0 并关闭 VMfile 流。

清单 1.1　头文件 ABSSTACK.H 的源代码

```
#ifndef _ABSSTACK_H
#define _ABSSTACK_H

#include <fstream.h>

/*----------------------------------------
Implements classes of generic stacks with the following set of operations:
      + Push
      + Pop
      + Clear
----------------------------------------*/

const unsigned MAX_STR = 80;
enum Boolean { false, true };

// ***** Asbtract Stack *****
class CAbsStack
{
  protected:
    unsigned nHeight;              // height of stack
    Boolean bAllocateError;        // dynamic allocation error

    CAbsStack() {};
    virtual ~CAbsStack() { };

    // ***** State Query Methods *****
    Boolean GetAllocateError() { return bAllocateError; }
    Boolean IsEmpty() { return (nHeight == 0) ? true : false; }

    // ***** Object Manipulation Methods *****
    virtual void Push(const char * szStr) = 0;
    virtual Boolean Pop(char * szStr) = 0;
    virtual void Clear() = 0;
};

struct StrStackRec {
    char szNodeData[MAX_STR+1];
    StrStackRec * pNextLink;
};

class CStrStack : public CAbsStack
{
  public:
    CStrStack();
    virtual ~CStrStack() { Clear(); }

    // ***** Object Manipulation Methods *****
    virtual void Push(const char * szStr);
    virtual Boolean Pop(char * szStr);
    virtual void Clear();
```

```
  protected:
    StrStackRec * pTop;      // pointer to the top of the stack
};

class CVMStrStack : public CAbsStack
{
  public:
    CVMStrStack(const char * Filename);
    virtual ~CVMStrStack() { Clear(); }

    // * * * * * State Query Methods * * * * *
    char * GetErrorMessage() { return szErrorMessage; }

    // * * * * * Object Manipulation Methods * * * * *
    virtual void Push(const char * szStr);
    virtual Boolean Pop(char * szStr);
    virtual void Clear();

  protected:
    char szDataBuffer[MAX_STR+1];        // data buffer
    char szErrorMessage[MAX_STR+1];      // error message
    fstream VMfile;                      // virtual stream handle
};

#endif
```

清单 1.2　库文件 ABSSTACK.CPP 的源代码

```
#include "absstack.h"
#include <string.h>

//------ CStrStack ------

CStrStack::CStrStack()
// constructor to initialize generic stack.
{
    nHeight = 0;
    bAllocateError = false;
    pTop = NULL;
}

//------ Push ------

void CStrStack::Push(const char * szStr)
// pushes the data accessed by STRING szStr onto the stack.
{
  StrStackRec * p;

  bAllocateError = false;
  if (pTop) {
      p = new StrStackRec;          // allocate new stack element
      if (! p) {
         bAllocateError = true;
         return;
      }
      strcpy(p->szNodeData, szStr);
      p->pNextLink = pTop;
      pTop = p;
  }
  else {
      pTop = new StrStackRec;
```

```
        if (! pTop) {
            bAllocateError = true;
            return;
        }
        strcpy(pTop->szNodeData, szStr);
        pTop->pNextLink = NULL;
    }
    nHeight++;
}

//------ Pop ------

Boolean CStrStack::Pop(char * szStr)
/* pops the top of the stack and returns a Boolean value. Function
    returns true if the operation was successful. A false value is
    returned if the Pop message is sent to an empty stack.
*/
{
  StrStackRec * p;

  if (nHeight > 0) {
      strcpy(szStr, pTop->szNodeData);
      p = pTop;
      pTop = pTop->pNextLink;
      delete p;                 // deallocate stack node
      nHeight--;
      return true;              // return function value
  }
  else
      return false;             // return function value
}

//------ Clear ------

void CStrStack::Clear()
// clears the generic stack object.
{
  char szStr[MAX_STR+1];

  while (Pop(szStr));
        /* do nothing */;
}

//------ CVMStrStack ----

CVMStrStack::CVMStrStack(const char * Filename)
// constructor to initialize generic stack.
{
  nHeight = 0;
  VMfile.open(Filename, ios::in | ios::out | ios::binary);

  if (! VMfile) {
    strcpy(szErrorMessage, "Cannot open file ");
    strcat(szErrorMessage, Filename);
    return;
  }
  else {
    bAllocateError = false;
    strcpy(szErrorMessage, "");
  }
}
```

```
//------ Push ------

void CVMStrStack::Push(const char * szStr)
// pushes the data accessed by parameter szStr onto the stack.
{
    nHeight++;
    VMfile.seekg((nHeight-1) * (MAX_STR+1));
    VMfile.write((unsigned char * ) szStr, MAX_STR+1);
}

//------ Pop ------

Boolean CVMStrStack::Pop(char * szStr)
// pops the top of the stack and returns a Boolean value.
{
  if (nHeight > 0) {
      nHeight--;
      VMfile.seekg(nHeight * (MAX_STR+1));
      VMfile.read((unsigned char * ) szStr, MAX_STR+1);
      return true;
  }
  else
      return false;
}

//------ Clear ------

void CVMStrStack::Clear()
// clears the generic stack object.
{
  nHeight = 0;
  VMfile.close();
}
```

让我们看看 ABSSTACK.CPP 库的测试程序。清单 1.3 包含 STACK1.CPP 程序，该程序可用来测试清单 1.1 中所声明的各种类。该程序声明了对象 aStack 和 aVMStack，它们分别是 CStrStack 及 CVMStack 的实例。这个程序完成如下相关工作：

(1) 实例化 aStack 和 aVMStack。该程序利用所支持的文件 VS.DAT 来建立后面的那个对象。

(2) 用一个 for 循环将字符串数组 pStringArray 中的元素压入堆栈 aStack。这个程序向对象 aStack 传送一条 Push 消息 (message) 以压入每个字符串。

(3) 从堆栈对象 aStack 中弹出字符串。该程序用一个 while 循环向 aStack 实例传送 Pop 消息，循环迭代的次数与将要弹出堆栈的字符串的数目一样。while 循环体显示出已弹出堆栈的每个字符串。

(4) 用一个 for 循环将字符串数组 pStringArray 中的元素压入堆栈 aVMStack。这个程序向对象 aVMStack 传送一条 Push 消息以压入每个字符串。

(5) 从堆栈 aVMStack 中弹出字符串。该程序用一个 while 循环向 aVMStack 实例传送 Pop 消息。循环迭代的次数与将要弹出堆栈的字符串的数目一样。while 循环体显示出已弹出堆栈的字符串。

STACK1.MAK 文件应该包含 ABSSTACK.CPP 和 STACK1.CPP 文件，并且该文件位于 \MSVC\VC 目录中。

清单 1.3　用于测试 ABSSTACK.H 中各种类的 STACK1.CPP 程序文件

```
/* Program to test stacks of strings */

#include "absstack.h"
#include <iostream.h>

main()
{
  const unsigned MAX_STRINGS = 10;
  char* Filename = "aVMStack.DAT";
  char* pStringArray[MAX_STRINGS] =
              { "California", "Virginia", "Michigan",
                "New York", "Washington", "Nevada",
                "Alabama", "Alaska", "Florida", "Maine" };

  char chAKey;
  char szString[MAX_STR+1];
  CStrStack aStack;
  CVMStrStack aVMStack(Filename);

  cout << "Testing heap-based stacks objects\n\n";
  for (int i = 0; i < MAX_STRINGS; i++) {
      cout << "Pushing ";
      cout.width(12);
      cout << pStringArray[i] << " into the stack\n";
      aStack.Push(pStringArray[i]);
  }

  cout << "\nEnter any character to continue... ";
  cin >> chAKey;
  cout << "\n";

  while (aStack.Pop(szString)) {
      cout << "Popping off ";
      cout.width(12);
      cout << szString << " from the stack\n";
  };

  cout << "\nEnter any character to continue... ";
  cin >> chAKey;
  cout << "\n\n\n\n";

  cout << "Testing virtual stacks objects\n\n";
  for (i = 0; i < MAX_STRINGS; i++) {
      cout << "Pushing ";
      cout.width(12);
      cout << pStringArray[i] << " into the stack\n";
      aVMStack.Push(pStringArray[i]);
  }

  cout << "\nEnter any character to continue... ";
  cin >> chAKey;
  cout << "\n";

  while (aVMStack.Pop(szString)) {
      cout << "Popping off ";
      cout.width(12);
      cout << szString << " from the stack\n";
  }
```