Visual C++

图文程序设计

顾晓明　韩复生　等编著

国防工业出版

Visual C++图文程序设计

Visual C++图文程序设计

顾晓明　韩复生 等　编著

国防工业出版社

·北京·

图书在版编目(CIP)数据

Visual C++图文程序设计/顾晓明,韩复生等编著.-北京:国防工业出版社,1996.3
ISBN 7-118-01533-4

Ⅰ.V… Ⅱ.①顾… ②韩… Ⅲ.C语言-程序设计,图文 Ⅳ.TP312C

中国版本图书馆CIP数据核字(95)第17032号

国防工业出版社出版发行
(北京市海淀区紫竹院南路23号)
(邮政编码 100044)
北京怀柔新华印刷厂印刷
新华书店经售
*
开本 787×1092 1/16 印张 30¾ 713千字
1996年3月第1版 1996年3月北京第1次印刷
印数:1—4000册 定价:41.50元

前　言

自 Windows 操作系统问世以来，以 Windows 为基础的各种软件工具和开发环境如雨后春笋，Visual C++就是其中之一。继 Microsoft 公司推出了支持 Windows 程序设计的 Microsoft C/C++7.0 编程环境之后，Borland 公司也推出了便于使用的 Borland C++ 3.1&4.0。目前，随着 Visual C++的问世，大量 Windows 程序员又开始以 Visual C++为开发平台进行程序开发工作。这些开发环境的推出，极大地方便了计算机图形、窗口和界面的制作，使程序员的编程效率成倍地提高。

Visual C++是目前深受用户欢迎的程序开发环境，原因在于它不仅支持 OOP 和 Windows 程序设计，而且还提供了下列实用工具：集成开发环境 Visual Workbench (VWB)、应用程序生成器 AppWizard、用来建立和编辑 Windows 资源的交互式图形工具 AppStudio、交互式类剖析工具 ClassWizard 和 Visual Workbench Browser 等。

当然，Visual C++最重要的特色还在于它提供了 MFC 类库，这是一个包含 100 多个类的集合，源代码超过 60 000 行，该库使 Visual C++软件工具工作得天衣无缝。本书就从程序员的角度出发集中讨论如何运用 MFC 类库中的类来编写 Windows 图文应用程序。至于上述实用工具的用法，则请参阅相关的 Visual C++ 文档。

本书主要讨论 MFC 中的通用类(字符串类、数组类、表类和映射类)、模态及非模态对话框数据转移类 CDataExchange、通用对话框类(文件选择对话框类 CFileDialog、字体选择对话框类 CFontDialog、颜色选择对话框类 CColorDialog、打印机设置对话框类 CPrintDialog、文本查找/替换对话框类 CFindReplaceDialog)、异常处理类(内存异常类 CMemoryException、文件异常类 CFileException、档案异常类 CArchiveException、非支持异常类 CNotSupportedException、资源异常类 CResourceException、用户异常类 CUserException 和 OLE 异常类 COleException)、各种绘图对象类和绘图类。这些类封装了开发一般程序所需的大量函数和变量，掌握了这些类的定义和用法，就为开发大型图文应用程序奠定了基础。在讨论每个 MFC 类时，我们先给出该类的成员函数及数据成员，再用程序实例演示该类的用法，最后对程序进行剖析。

此外，本书还介绍了如何用 TEMPLDEF 实用工具来创建模板文件、各种内存管理函数的用法以及与图形程序设计有关的各种技术。TEMPLDEF 实用工具有助于提高开发程序的效率，内存管理函数的恰当使用有助于提高程序的运行效率，而绘图技术则是开发 Windows 应用程序所不可缺少的。

在讨论图形程序设计技术时，我们给出了几个典型例子，例如根据给定的函数来绘制曲线，用不同的颜色来填充不同的几何区域，使用户能够借助鼠标绘制各种图形。书中部分程序是基于 Windows 的窗口图形程序，少量程序是文本程序，所以读者应具有 C++语言编程经验，并对 Windows 编程有一定的了解。如果读者对 C 语言

环境不太了解，则请在阅读本书之前先参考其它相关书籍。

值得指出的是，Visual C＋＋是一个庞大而复杂的编程环境，限于篇幅，本书不可能囊括其全部内容，但是，本书尽量做到“以点带面”，力求挖掘 Visual C＋＋的精髓。

本书由多人合作而成，尽管我们在编写过程中十分谨慎，并对书中的程序进行了调试，但由于时间有限，书中可能还有不足之处，恳请读者指正。

编　者

内 容 简 介

Visual C++是Windows操作系统中用来开发OOP程序的编程环境。本书以介绍Visual C++ MFC类库为线索，集中讨论如何用Visual C++开发Windows图文应用程序，主要内容包括：通用类、模板文件、对话框数据转移类、对话框类、异常处理类、内存管理机制、与图形有关的类以及各种图形程序设计技术。在前面列出的各种C++类中，有些类是Visual C++系统本身提供的，对于这些类，本书给出了完整的定义，包括数据成员和成员函数的定义及使用说明；有些类是作者为了实现某种特殊功能而补充的，对于这些类，本书给出了基类以及扩充类的定义，并对扩充类进行了剖析。

书中包含大量实例程序，它们说明了各种Visual C++类的用法，其中少量实例是QuickWin应用程序，而大部分实例是使用了MFC类库的Windows应用程序。这些程序演示了Windows系统中Visual C++的图形、文本和界面功能，读者可实际运行这些程序，并将运行结果与本书提供的输出画面进行比较，以便快速掌握Visual C++程序设计原理和技术。另外，对书中的程序稍作修改，即可在Microsoft C/C++环境中运行。

本书内容全面，实例丰富，可供C++（尤其是Visual C++）语言程序员参考，也可作为有关Windows程序设计方面的培训教材。

目　录

第一章　Visual C++类库中的通用类

用 Visual C++为 IBM 个人计算机及 PC 兼容机开发 Windows 应用程序与用其他 C++语言开发文本程序一样方便，甚至更加方便，因为 Visual C++中包含一套功能强大的、预先编写好的类库 MFC(Microsoft Foundation Class)，它可以极大地简化 Visual C++的编程工作。借助 Visual C++ MFC 类库提供的类和 Visual C++的交互工具，很容易就能生成一个可以实际运行的 C++应用程序。所有要做的就是选择菜单和对话框，告诉工作台(Workbench)希望生成哪一类应用程序，然后用 VWB 编辑器或其他 Visual C++工具编写应用程序代码。

MFC 库中包含用以实现数组和表的类以及基于杂凑表的映射(map)类。另外，MFC 库还声明了一个用来模拟字符串对象的类。这些类不仅节省了编程时间，而且能有效地为开发 Windows 应用程序提供支持。本章将介绍各种常用类。

1.1　字 符 串 类

MFC 库实现 CString 类是为了模拟字符串对象，这些字符串对象与 BASIC 中的字符串类似。事实上，CString 类的某些成员函数是以内部 MS—BASIC 函数为基础的。头文件 AFX.H 中包含 CString 类的声明，如清单 1.1 所示。

清单 1.1　头文件 AFX.H 中声明的 CString 类

```
class CString
{
public:
//Constructors
  CString();
  CString(const CString &. stringSrc);
  CString(char ch, int nRepeat = 1);
  CString(const char * psz);
  CString(const char * pch, int nLength);
#ifdef _NEARDATA
// Additional version for far string data
  CString (LPCSTR lpsz);
  CString (LPCSTR lpch, int nLength);
#endif
  ~CString();
```

```
// Attributes & Operations:as an array of characters
  int GetLength() const;
  BOOL ISEmpty() const;
  void Empty() const;                    // free the data
  char GetAt(int nIndex) const;          // 0—based
  char operator[](int nIndex) const;     // same as GetAt
  void SetAt(int nIndexm char ch);
  operator const char * () const;        // as a C String

// overloaded assignment
  const CString & operator=(const CString & stringSrc);
  const CString & operator=(char ch);
  const CString & operator=(const char * psz);

// string concatenation
  const CString & operator+=(const CString & string);
  const CString & operator+=(chat ch);
  const CString & operator+=(const char * psz);

  friend CString AFXAPI operator+(const CString & string1,
                                  const CString & string2);
  friend CString AFXAPI operator+(const CString & string, char ch);
  friend CString AFXAPI operator+(char ch, const CString & string);
  friend CString AFXAPI operator+(const CString & string,
                                  const char * psz);
  friend CString AFXAPI operator+(const char * psz,
                                  const CString & string);
// string comparison
  int Compare(const char * psz) const;          // straight character
  int CompareNoCase(const char * psz) const;    // ignore case
  int Collate(consr char * psz) const;          // NLS—aware

// simple substring extraction
  CString Mid(int nFirst, int nCount) count;
  CString Mid(int nFirst) const;
  CString Left(int nCount) const;
  CString Right(int nCount) const;
  CString SpanIncluding(const char * pszCharSet) const;
  CString SpanExcluding(const char * pszCharSet) const;

// upper/lower/reverse conversion
  void MakeUpper();
  void MakeLower();
  void MakeReverse();

// searching (return starting index, or -1 if not found)
// look for a single character match
```

```
  int Find(char ch) const;                    // like C strchr
  int ReverseFind(char ch) const;
  int FindOneOf(const char * pszCharSet) const;

// look for a specific sub-string
  int Find(const char * pszSub) const;        // like C strstr

// input and output
#ifdef _DEBUG
  friend CDumpContext & AFXAPI operator<< (CArchive & ar,
                                          const CString & string);
#endif
  friend CArchive & AFXAPI operator<< (CArchive & ar,
                                       const CString & string);
  friend CArchive & AFXAPI operator>> (CArchive & ar,
                                       CString & string);

// Windows support
#ifdef _WINDOWS
  BOOL LoadString(UINT nID);          // load from string resource
                                      // 255 chars max
// ANSI<->OEM support (convert string in place)
  void AnsiToOem();
  void OemToAnsi();
#endif  //  _WINDOWS

// Access to string implementation buffer as C character array
  char * GetBuffer(int nMinBufLength);
  void ReleaseBuffer(int nNewLength = -1);
  char * GetBufferSetLength(int nNewLength);

// Implementation
protected:
// length/size in characters
// note: an extra character is always allocated
  char * m_pchData;                   // actual string (zero terminated)
  int m_nDataLength;                  // does not include terminating 0
  int m_nAllocLength;                 // does not include terminating 0

// implementation helpers
  void Init();
  void AllocCopy(CString & dest, int nCopyLen, int nCopyIndex,
                 int nExtraLen) const;
  void AllocBuffer(int nLen);
  void AssignCopy(int nSrcLen, const char * pszSrcData);
  void ConcatCopy(int nSrc1Len, const char * pszSrc1Data,
                  int nSrc2Len, const char * pszSrc2Data);
```

```
    void ConcatInPlace(int nSrcLen, const char * pszSrcData);
};
// Compare helpers
BOOL AFXAPI operator== (const CString & s1, const CString & s2);
BOOL AFXAPI operator== (const CString & s1, const char * s2);
BOOL AFXAPI operator== (const char * s1, const CString & s2);
BOOL AFXAPI operator! = (const CString & s1, const CString & s2);
BOOL AFXAPI operator! = (const CString & s1, const char * s2);
BOOL AFXAPI operator! = (const char * s1, const CString & s2);
BOOL AFXAPI operator< (const CString & s1, const CString & s2);
BOOL AFXAPI operator< (const CString & s1, const char * s2);
BOOL AFXAPI operator< (const char * s1, const CString & s2);
BOOL AFXAPI operator> (const CString & s1, const CString & s2);
BOOL AFXAPI operator> (const CString & s1, const char * s2);
BOOL AFXAPI operator> (const char * s1, const CString & s2);
BOOL AFXAPI operator<= (const CString & s1, const CString & s2);
BOOL AFXAPI operator<= (const CString & s1, const char * s2);
BOOL AFXAPI operator<= (const char * s1, const CString & s2);
BOOL AFXAPI operator>= (const CString & s1, const CString & s2);
BOOL AFXAPI operator>= (const CString & s1, const char * s2);
BOOL AFXAPI operator>= (const char * s1, const CString & s2);
```

CString 类声明了一组构造函数、一个析构函数、各种成员函数和一组友元函数。下面几小节将分组讨论大部分不同的成员函数和友元函数。

1.1.1 构造函数

CString 类声明了七个构造函数，其中两个提供远程字符串数据。这七个构造函数中含有缺省(default)和拷贝(copy)构造函数，而其他构造函数通过使一个字符重复指定的次数或用一个 ASCII 字符串建立 CString 类的实例。

下面是建立 CString 类实例的例子：

```
CString s1;                    // string object is empty
CString s2("Hello World!");    // contains "Hello World!"
CString s3(s2);                // also contains "Hello World!"
CString s4('A');               // contains "A"
CString s5('A', 3);            // contains "AAA"
```

1.1.2 属性函数

CString 类包含如下成员函数，这些函数返回一个实例的普通属性。

(1) 成员函数 GetLength：它返回存储在一个实例中的字符的数目。

(2) 布尔成员函数 IsEmpty：如果对象实例中不包含字符，则该函数返回 TRUE，否则返回 FALSE。

(3) 成员函数 Empty:它清空一个实例并释放内存。

下面是使用上述成员函数的例子:

```
CString Str ("0123456789");
cout << "String has " << Str. GetLength() << "characters\n";
Str. Empty();

if (Str. IsEmpty())
  cout << "String is now empty\n";
else
   cout << "Looks like function Empty goofed! \n";
```

1.1.3 访问函数

CString 类提供了如下成员函数,以访问一个 CString 对象中的单个字符或整个字符串。

(1) 成员函数 GetAt 和操作符[ ]:它们取出下标参数 nIndex 处的字符。在 CString 类中,字符的下标从 0 开始。

(2) 成员函数 SetAt:它在一个确定的下标处存放一个字符。

(3) 无参数操作符 const char * :它访问存储在一个 CString 实例中的 ASCII 字符串。

(4) 成员函数 GetBufter,ReleaseBuffer 和 GetBufferSetLength:这些函数能够在一个 CString 实例中存储字符,它们有助于设置一个 CString 实例作为 char * 参数的值。

下面是使用上述函数和操作符的例子:

```
CString Str ("Hello");
char szStr[81] = "HELLO";
cout << "Strings '" << szStr << "' and '"
     << (const char *) Str << "' are ";

if (strcmp(szStr, (const char *) Str) ! =0)
  cout << "not";
cout << "equal";

// replace each character with a !
for (int i = 0; i < Str. GetLength (); i++)
    Str. SetAt (i, '!');
cout << "String is now '" << (const char *) Str << "' \n";

// replace each character with an @
cout << "String is now '";
for (i = 1; i < Str. GetLength(); i++) {
    Str[i] = '@';
    cout << Str [i];
}
cout << "'\n";
```

函数 GetBuffer 返回一个 char * 指针，它指向 CString 实例中的字符串。可以直接利用函数结果或使用另一条语句将结果赋给一个 char * 指针，从而使指针指向一组新字符。一旦完成了这项工作，就必须在传送其它任何消息之前向 CString 实例传送一条 ReleaseBuffer 消息。

函数 GetBuffer 带一个参数 nMinBufLength，这个参数确定了 CString 字符缓冲区的最小尺寸(可能排除了空终结符)。函数 ReleaseBuffer 带一个参数 nNewLength，该参数的缺省值为－1，它确定了除空终结符外的字符串的新长度。取缺省值时，该函数将 CString 实例的大小设置成当前字符串长度。函数 GetBufferSetLength 返回一个指向 CString 实例字符串的指针，并且将字符串的大小设置成由参数 nNewLength 确定的数值。该函数可以根据参数 nNewLength 的值扩展或缩小缓冲区。

下面是一个使用函数 GetBuffer 和 ReleaseBuffer 的例子：

```
const MAX = 80;
CString Str;
char * pszStr = Str.GetBuffer (MAX+1);
cout << "Enter your name: ";
cin.getline (pszStr, MAX);
Str.ReleaseBuffer (-1);
cout << "\nHello << (const char *) Str << "\n";
```

1.1.4 赋值操作符

CString 类声明了许多操作符，以赋值和连接字符串及字符。类中有三个“＝”操作符，它们分别将 CString 实例、ASCII 字符串或字符赋给 CString 的实例。类中还有类似的“＋＝”操作符，它们用来将三种类型的项连接到 CString 类实例上。

1.1.5 连接操作符

CString 类声明了 5 个“＋”友元操作符，它们可用来将 CString 类实例与其他 CString 类实例、ASCIIZ 字符串或字符连接在一起。C＋＋编程规则规定，一个“＋”操作符的两个参数之一必须是友元类。

下面是使用操作符“＝”，“＋＝”和“＋”的例子：

```
CString Str1 = "Hello";
CString Str2 = "How";
CString Str3 = "you";
CString Str4;
Str4 = Str1 + "there!" + Str2 + "are" + Str3 + '?';
// next statement displays "Hello there! How are you?"
cout << (const char *)Str4 << "\n";
Str4.Empty();                    // clear string
Str4 += Str1;
Str4 += "there!"
```