Visual C++
程序开发指南

DISK
INCLUDED

· 学习使用 Visual C++的新特性和增强特性，包括 OLE 2.0 和 Microsoft Foundation Class

· 随附软盘，提供众多用于 C++ 对象类的完整源代码，用以加速 Windows 环境的图形软件开发

[美]Alex Leavens 著
方宇炜 田学锋 译
王 勇 校

電子工業出版社
Publishing House of Electronic Industry

PHEI M&T BOOKS

Visual C++程序开发指南

[美] Alex Leavens 著

方宇炜 田学锋 译

王 勇 校

电子工业出版社

1994.7

(京)新登字 055 号

内 容 提 要

Visual C++ 1.5 是美国微软公司(Microsoft Corp.)新近推出的新一代程序设计语言,可用于 16 位及 32 位环境下的程序设计。《Visual C++程序开发指南》适用于在 Windows 环境下学习和使用 Visual C++进行程序设计的所有人员。书中以大量的实例介绍了 Visual C++ 1.5 版本的新特点和增强功能,并介绍了众多优秀的程序设计技术和风格,反映了现代程序设计技术的风貌。与本书一起提供给读者的软盘,包含了书中所有范例的源代码和通用图形库的完整源代码。

本书适合所有 Windows 环境下的程序设计人员和开发人员,对大学计算机专业从事科研工作的研究生也有很高的参考价值。

程序开发指南
〔美〕Alex Leavens 著
方宇炜 田学锋 译
王勇 校
责任编辑 王世忠

*

电子工业出版社出版(北京市万寿路)
电子工业出版社发行 各地新华书店经销
志达公司排版印刷部排版
北京市顺义县天竺颖华印刷厂印刷

*

开本:787×1092 毫米 1/16 印张 20.75 字数 488 千字
1994 年 7 月第 1 版 1994 年 7 月第 1 次印刷
印数:10100 册 定价:68.00 元(含磁盘)
ISBN7-5053-2635-X/TP·806

本书的适合读者

如果您现在是一个C或C++语言的编程人员，并且想进一步学习Viusal C++语言(1.5版本)，那么这本书非常适合于您。如果您是一个C++的编程人员，并且正在或打算在Windows环境下编程，那么，《Visual C++程序开发指南》对您非常适合。

另外，Windows环境的开发人员还可以学到一些以不可思议的方法处理系统的专业应用软件。开发人员将发现，为一种应用程序服务的消息是如何被解释和改变为另外一种应用程序服务的。

对于任何一个打算使用C++的公司，《Visual C++程序开发指南》将告诉他们如何建立通用类的Windows应用程序，如开发和维护Windows的中继(hooks)，建立带多种视窗的多个文档，并且定义Windows的中继，建立多视窗，以及定义用于应用程序间通信的系统消息。

《Visual C++程序开发指南》以对话的语气使您了解和掌握1.5版本的新特点和增强功能。例如：数据库应用软件的新型Microsoft Foundation Class支持，开发和支持OLE2.0的地方编辑(place editing)、全开放式编辑、剪贴板、粘贴以及OLE目标链接、拖放直观编辑。另外，与本书一起提供给读者的磁盘中包含了所有应用程序范例的源代码以及通用图形库(Universal Graphics Library)的完整源代码。

引　言

在计算机时代以前，编程人员（其实不存在编程人员）不怎么关心如何开发程序。后来，当最原始的计算机得到发展以后，最让编程人员棘手的问题是死在巨大真空管里的昆虫，即大量程序中存在的各类错误，编程人员们试图对这些真空管进行控制，同时也通过这些真空管束发泄他们的愤恨，因此出现了 bug（错误）这个词。

在这一时期，程序就象是一只双手得以进化的野兽，在任何时候都在变化，并且相当低级。后来，当计算机变得更加复杂时，程序也相应变得复杂了，虽然磁盘空间仍然是非常昂贵。

在过去的几十年中，我们发现计算机在速度、功能以及复杂程度（包括可利用的空间）上有了巨大的发展，但是编程人员们仍然以原始的方式编写程序——通过手完成录入，令人非常痛苦，并且每次只执行一步。

为什么

部分原因是因为大部分编程人员不知道有什么更好的办法。虽然计算机已经出现将近半个世纪了，但一般的编程人员，编写程序的时间只有几年。他或她可能是刚从学校毕业，或从其它领域改行为编程。不管什么原因，一般的编程人员编写程序只有三年左右的时间。在以后的另外三年时间里，这个编程人员可能升迁（成为一名管理或销售人员）。这意味着编写程序的时间一般为六个月至六年，随着记忆流逝，这并不是一段很长的时间。

而我们需要的是 15 年的老手，一些不让他们的技术退化的编程人员，带有魔力的精灵，以及“长有白胡子、带有粗哑噪音的干瘪老头”。

我还没有老到那种程度

然而我也有些年头了。我编程已有 20 年了，其中的 16 年是专业人员。如此丰富的经验让我注意到一些问题。首先，很多编程人员的确在以 20 年前的方法工作。语言已经变了，界面变得非常完美，且有强大功能的调试程序，但是他们仍然按下面的方式编写程序。

1. 写出一些代码。
2. 编辑。
3. 看是否行得通。
4. 找出错误之处。

当然，上面的某些步骤是不可避免的。编程人员每次只能写一行程序。然而，我们可以让编程过程变得不是如此痛苦，这得依靠手工技巧。我们有一些工具和技术，可以使编程人员的生活变得非常轻松。在这本书中，将介绍一些这样的工具和技术。

目 录

第一章　程序设计——简单历史和风格指南

编程是一项令人讨厌的工作，但是有人不得不去做它，并且有时他就是您。在关键时刻，老板能够指望谁呢？

然而，这并不意味着您一直坐在那儿进行编程，有许多东西能使您的编程变得非常轻松，其中之一就是开发一种适当的结构来改进编程。

1.1　第一步：获得一个系统

我注意到，大部分编程人员没有任何系统来支持他们的工作，他们只是以旧的方式将代码堆积在一起而已：函数原型被随意地放置在源文件开头，定义没有特殊的顺序，没有使用一组匹配的#ifndefs 进行进一步定义等等。

这样做不好！

我教学生的第一样东西就是我在许多年中逐步形成的一整套编程标准以及方法，这并非我一个人特有，而是许多人经过许多年才形成的。

我的编程约定实际上是指风格，大部分是使用简单方法使编写的代码更加美观。除了清晰美观外，另外一个原因是，这样做很容易发现代码中的错误。

据我们所知，许多编程人员极力使他们的代码看上去难懂：很保密且不易破译。而我（我也鼓励我的学生）却力求清晰、简单优雅的代码。毕竟是代码所包含的算法而不是代码本身，才能显示编程人员的聪明。

我的许多编程约定是许多年的经验，利用这些技术可以避免一整类错误。例如，在每个头文件的开始使用适当的#ifndef 和#define directives，可以避免多个头文件的错误。

这里包括了我的整个资料，并在需要帮助的地方进行了说明。

1.2　ShadowCat 技术编码约定

1.2.1　简　　介

这部分内容讲述了使用 ShadowCat 技术编写 C 程序的编程约定。这里，我假定读者熟悉 C 程序。

1.2.2　命名和注释标准

这部分内容反映 C 和 C++中最新的命名和注释标准。特别地，注释符已被更改以反映 Microsoft 规范（//，而不是/*），这已被加入 C++的约定中。

对于 C 语言编辑器而言，Microsoft 注释风格//不是 ANSI 标准，如果想把程序移入其它只有 ANSI C 编辑器的平台（例如 UNIX），那么仍需使用旧的注释风格/*和*/。

1.3 函　数

1.3.1 函数风格约定

函数应该使用如下方式编写：

- 每个函数以换页符(^L)开始，这样，在打印每个函数时可将其打在一页的开始。
- 每个函数应该遵循如下的通用格式。

(NB-Tab 终止是 4 个空格而不是 8 个)[ShadowCat]
通用的函数头格式：

```
//----------------------
//
// FunctionName()
//   Description of the function's purpose
//
// Returns:
//  what the function returns (and why)
//
// Assumptions:
//  Any assumptions about the programming universe that
//  the function lives in which are important and/or non-obvious.
//
//  'The variable gwHand contains the global handle to our window
//   data structures which contain the handles to currently open
//   windows.'
//
//
   void WINAPI
FunctionName ( VAR       var1,     // Short description of 1st variable
               VAR       var2,     // Short description of 2nd variable
               etc. )              // More variable descriptions here.
```

除了 FunctionName()这一项外，其它部分(Parameter、Returns、Assumptions)可以省略，而且这样做没有省略关键性的信息。例如，若函数没有返回值，应定义为 void，此时“Returns”项可以省略(虽然不推荐这样做)。然而，一般建议列出函数头的每一项。

函数头实例：

```
//-----------------
//
// DrawCircleOnScreen()
//  This routine is responsible for actually drawing a circle on the
// screen in response to some input from some other part of the program
// (ie, a user input, or a system redraw call).
//
//
// Returns:
//   Nothing
//
// Assumptions:
//   This routine assumes that it can get a colored pen to draw with
// from a global array of pens, hPenList[], which must contain handles
// to the list of pens that are currently available. There must be
// AT LEAST ONE pen available, or the routine will choke.
//
  void WINAPI
DrawCircleOnScreen ( HWND  hWnd,          // Window handle to draw into
                     HDC   hDC,           // DC to draw with
                     int   x,             // X coord of center of circle
                     int   y,             // Y Coord of center of circle
                     WORD  rad )          // Radius of circle (ALWAYS
                                          // POSITIVE)
{
  // Code here...
}
```

1.3.2　函数定义格式

所有函数必须说明，即使函数不返回任何值，它也必须说明为 void。Microsoft Windows 程序对此有特别规定，不仅要说明函数的类型，而且要说明函数是 NEAR 或 FAR，是 PASCAL 或 C(对于 Windows 程序，推荐使用源于 Windows 3.1 SDK 的 WINAPI，因为这样做可使程序在各种平台和存储模式之间有更大的可移植性)。下面是函数定义的例子：

```
//-----------
//
// FunctionHeader...()
//
  void WINAPI
```

```
DrawCircleOnScreen ( HWND  hWnd,          // Window handle to draw into
                     HDC   hDC,           // DC used for drawing
                     int   x,             // X Coord of center of circle
                     int   y,             // Y Coord of center of circle
                     WORD  rad )          // Radius of circle (ALWAYS
                                          // POSITIVE)
```

这里，调用被定义为一个 void WINAPI 调用。

函数的每个参数是在调用表中定义而不是在其下面定义，这与 ANSI C 准则是一致的。在参数表中，不应有与函数调用无关的参数说明。另外，参数表的每一个变量各占一行并且使用制表符使行与行之间头对齐。如果变量需要说明，那么说明应该位于同一行变量的后面。说明可以占好几行，但必须与第一行的说明对齐，例如：

```
FunctionName ( VAR       var1,    // Really long and involved comment about
                                  // what this variable is and why it is
                                  // so very important!!
FunctionName ( type      var1,     // Comment
               type      var2,     // Comment
               etc..... )
{
  internal variable list;      /* Comments on internal
                                  // variables should follow
                                  // variable names

  //-----------------  <-Variable and code boundary
                       marker (also used in
                       temporary code segment
                       areas). The boundary
                       marker consists of a
                       leading blank line,
                       the 'dash comment' and
                       a trailing blank line.
  //
  // If a comment for an assignment or other small piece of
  // code is lengthy, it should precede the code in this format.
  //
  //
  // Assignment statements, conditionals, etc., should be coded
  // with whitespace for legibility.
  //
```

```
variable=assignment;    // Illegible code....
variable = assignment;  // Legible code
                        // Short comment style
                        // Alternative comment style for
                        // medium length comments.
                        // (Discouraged)
//
// Braces for conditional expressions go on separate
// lines than the conditional expression itself. The
// closing brace should have a comment indicating what
// expression it is closing. The one exception to this rule
// is where the closing brace is 'near enough' to the opening
// brace so that it is 'obvious' that the two braces are paired.
// Because code can change, however, it is recommended that the
// programmer always put the delimiting commert in.
//
if (condition)
{
  .
  .
  .
  Some code here....
  .
  .
  .
       while(foo)
       {
         ...Small amount of code...
       } // Note lack of definition of parent 'while(foo)'
  .
  .
  .
  More code here....


       } // End of 'if (condition)
       //
       // Note the use of a variable list/code section delimiter
       // in this local variable area.
```

```
//
while(condition)
{
  int i;                // Loop variable (see 'Naming Conventions')
  int count;            // Count of # of loop executions
                                                    <-leading blank line
  //-----------------
                                                    <-trailing blank line
  count = 0;
  for (i = 0; i < 10; i++)
  {
     count++;
  } // End 'for (i = 0; i < 10; i++)'
} // End 'while(condition)'
//
// Switch statements should also have the braces on
// separate lines for clarity. Alternatively, the opening brace
// can be on the same line as the switch, while the closing brace
// is on a separate line, although for consistency, this is
// discouraged.
//
// The default case should always be present, even if
// unused. The error handling macro ASSERT should be used in this
// case. When DEBUG is #defined, the ASSERT macro will generate
// error trapping code.
//
switch(foo)
{             <<-- Bracket is on following line, directly underneath
switch
      case XXXX:      // case: statements should be preceeded and
                      // followed by at least one line of whitespace.
                      // This is the preferred format of 'case:'
                      // comments.
        break;
      default: // Default case should never execute in this instance.
        ASSERT("Default case executed in switch(foo) of FunctionName()");
        break;
} // End 'switch(foo)'
```