Visual C++

内附范例光盘

程序设计技巧与实例

许福 舒志 张威 编著

- 本书的重点不是阐述编程原理，而是以实例的形式阐述VC编程中涉及的方法和技巧
- 编排原则由易到难，由简到繁，先介绍有关知识和技术，再以实例说明其开发应用
- 实例详尽、讲述通俗易懂，既可以作为入门者的教材，也可以作为中高级用户的参考书

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

技巧与实例丛书

Visual C++程序设计技巧与实例

许福 舒志 张威 编著

中国铁道出版社
2003·北京

（京）新登字 063 号

内 容 简 介

全书共分 14 章，主要内容包括 C/C++基础、VC 集成开发环境、对话框和控件、进程与线程、消息处理、文件和系统操作、文档/视结构、图形图象编程、多媒体编程、ActiveX、调试技术、数据库、网络编程、帮助制作等。

本书与以往讲述编程原理的书不同，本书的重点不是阐述编程原理，而是以实例的形式阐述VC编程中涉及的方法和技巧。因此章节之间以及章节之内的各小节之间的连贯性不是很强，基本上每一小节都自成体系，用来说明一个或者几个相关的编程技巧。

本书实例梯度比较明显，基本上每一章节的前半部分实例稍微简单些，后面的实例综合度比较大，稍微麻烦些，但每一个实例都给出了详细的操作步骤，因此无论是刚入门的新手，还是有相当编程经验的老手，阅读起来应该都不成问题。对于新手可以把这本书作为一本教材，系统地学习 VC 编程方面的相关知识；对于老手，可以把本书当作一本参考书，随时参阅。

图书在版编目（CIP）数据

Visual C++程序设计技巧与实例/许福，舒志，张威编著. —北京：中国铁道出版社，2003.3
（程序设计技巧与实例系列丛书）
ISBN 7-113-05173-1
Ⅰ.V… Ⅱ.①许…②舒…③张… Ⅲ.C 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2003)第 019965 号

书　　名：Visual C++程序设计技巧与实例
作　　者：许 福 舒 志 张 威
出版发行：中国铁道出版社（100054，北京市宣武区右安门西街 8 号）
策划编辑：严晓舟 魏 春
责任编辑：苏 茜 王占清
封面设计：孙天昭
印　　刷：河北省遵化市胶印厂
开　　本：787×1092 1/16 印张：26 5 字数：635 千
版　　本：2003 年 4 月第 1 版 2003 年 4 月第 1 次印刷
印　　数：1～5000 册
书　　号：ISBN 7-113-05173-1 /TP・907
定　　价：46.00 元

前　言

Visual Studio 6.0 是微软（Microsoft）的核心产品之一，而 Visual C++ 6.0 又是这款核心产品中的拳头产品。自 Visual C++上市以来，凭借其优异的品质，已成为众多 Windows 程序开发者的首选利器。

❑ Visual C++ 6.0 的优势

选择 Visual C++ 6.0 是您工作效率的保证，与其他的开发工具相比，它有如下几个特点：

(1) 拥有强大的编辑环境。Visual C++拥有一个强大的集成开发环境（IDE），工程的编辑、编译、调试都在这一环境下进行，用户不必在各个环境间切换，大大提高了开发效率。

(2) 拥有强大的类库 MFC 的支持。MFC 已成为 Windows 系统开发中最重要的类库之一，其类库丰富强大，是开发 Windows 程序的首选类库。

(3) 拥有强大的调试环境。在 Visual C++ 6.0 中可以设置断点、可以查看运行栈、查看相应的变量信息等，而且对程序员常犯的内存使用错误，Visual C++也会给出相应的警告，这些措施大大提高了调试的效率。

(4) 拥有较强的底层控制能力。与 Visual Basic 和 Delphi 等快速开发工具不同，Visual C++有很强的底层控制能力，Visual C++现在已经成为驱动程序开发的首选工具。

(5) 拥有强大的帮助系统 MSDN 的支持。MSDN 中含有几 GB 的文档和源代码，这里你几乎可以找到 Visual C++开发中遇到的任何问题的解决方案。

(6) 拥有一个高效的编译器，产生的可执行文件体积小巧、运行速度快。Visual C++ 6.0 由于对编译器进行了大量的优化，最后产生的可执行文件运行速度很快，而且很小巧。相比 Visual Basic 、Delphi 等工具，同类的代码 VC 生成的可执行文件基本上都是速度最快的。

(7) Visual C++ 6.0 还有其他一些优秀的品质：比如强大的 AppWizard 支持，方便强大的 ClassWizard 等。

赛门铁克（Symantic）公司的首席设计师曾经说过一句话："聪明的程序员用 Delphi，优秀的程序员用 Visual C++"，这句话很好地概括了 Visual C++的地位与作用。

❑ 本书的读者对象

本书主要是面对那些对于 Visual C++的基本操作、命令和各种工具的使用已经有一定了解的读者，要求读者有较强的 C 语言功底，本书实例详尽、讲述通俗易懂，既可以作为入门者的教材，也可以作为中高级用户的参考书。

❑ 本书的特点和章节安排

本书主要以实例的形式讲述 Visual C++ 6.0 的相关编程技巧，章节的安排如下：

第 1 章　C/C++基础

本章主要介绍 C/C++的基础知识，对于初学者需要仔细研读这部分内容，以便为将来的学习打下基础，对于 C/C++基础知识比较扎实的用户，可以略过这部分内容，直接进入后续

章节的学习。

第 2 章 MSDEV 集成环境与 VC 编译器

本章主要讲述集成开发环境（IDE）以及 VC 编译器方面的相关技巧。

第 3 章 对话框和控件

对话框和控件在 VC 编程中的使用频率很高，用好对话框和控件是学好 VC 编程的前提，本章主要讲述对话框和控件使用方面的相关技巧。

第 4 章 进程和线程

进程和线程是程序员必须掌握的核心概念之一，本章对于线程同步、进程和线程的优先级等知识进行了实例剖析，着重讲述进程和线程使用方面的相关技巧。

第 5 章 消息映射与处理

消息机制是 Windows 程序的核心之一，没有消息，用户界面与用户就无法进行交互，整个系统也无法运转，本章针对常见的鼠标、键盘、字符消息等进行了论述，同时也以实例的形式深入地讲述了这些方面的相关技巧。

第 6 章 文件和系统操作

文件操作在编程中使用频率很高，本章中以实例的形式阐述了 VC 文件操作中的技巧。

系统的一些参数有些时候对程序的运行起至关重要的作用，本章也以实例的形式阐述了这方面的一些内容。

第 7 章 窗口和桌面系统

本章阐述单文档(SDI)、多文档(MDI)、Explorer、文档/视结构方面的相关技巧。

第 8 章 图形图像编程

本章主要讲述设备环境（DC）、各种图形的绘制、图像的显示及处理等方面的内容。

第 9 章 多媒体技术

本章讲述剪贴板使用、位图动画、播放 wav、mid、cd、avi 等各种格式的文件以及如何制作屏保程序等方面的内容。

第 10 章 ActiveX

本章介绍 ActiveX 控件相关编程技巧，并实际制作了一个 ActiveX 控件——牌九 ActiveX 控件。

第 11 章 调试技术

本章介绍 VC 的调试环境、调试方法以及各种调试工具的使用。

第 12 章 VC 数据库编程

本章中介绍如何实现 VC 的数据库编程，并以实例讲解了通过 ODBC、DAO、OLE DB、ADO 等各种操作数据库的方法。

第 13 章 网络编程

本章介绍 VC 网络编程的相关技巧。

第 14 章 帮助系统

帮助是软件的一个重要组成部分，本章以两个实例讲解了传统的 hlp 格式的帮助文件以及现在流行的 chm 格式的帮助文件的制作方法。

❑ 本书作者群

本书由许福、舒志、张威主编完成，其中1、2、3、9、10、13、14章由许福编写，4、5、6、7、8、11、12章由舒志编写。此外，樊辉、杜群、淡学良、陈悦、陈武峰、朱禾、曾胜德、谭志、覃永、石军、陆庆、陈杰、林涛、欧贤能、莫海、桂松、秦卫、梁世康、彭力、董征、陈欢、杜强、唐春鸿、詹明、韦佳、秦彬、林波、石林、钱琳、陈锋、苏健、陈贤淑、陈晓娟、廖康良等同志在整理资料和排版方面给予了作者很大的帮助，在此，一并致以感谢。

鉴于编者水平有限，书中难免有不当之处，希望读者不吝赐教，我们会在适当的时间进行修订与补充，并发布在天勤网站：http://www.tqbooks.net“图书修订”栏目中。

编者

2003.2

目　录

第 1 章　C/C++基础 1

1-1　两个类互为成员如何声明 2
1-2　inline 函数使用问题 2
1-3　inline 和 static 有何区别 3
1-4　THIS_FILE 表示什么意思 3
1-5　try 和 TRY 有何区别 3
1-6　VC 中的函数调用习惯 4
1-7　用 C 语言实现参数个数可变的函数 8
1-8　创建和访问环境变量 9
1-9　常用的排序算法 11
1-10　二叉排序树的相关操作 20

第 2 章　MSDEV 集成环境与 VC 编译器 29

2-1　如何快速格式化代码块 30
2-2　如何调试 Release 版本程序 30
2-3　加快链接的小技巧 30
2-4　检测程序中的括号是否匹配 30
2-5　定位预处理指定 31
2-6　查看一个宏（或变量、函数）的宏定义 31
2-7　如何干净地删除一个类 31
2-8　如何让控制台应用程序支持 MFC 类库 31
2-9　如何汉化只有可执行代码的.exe 文件 31
2-10　自动提示出问题怎么办 31
2-11　如何将一个工程中的部分资源加到另一个工程中 32
2-12　一个经典的编译错误 32
2-13　VC6.0 对 VC5.0 的兼容性问题 32
2-14　VC 的 REMOTE DEBUG 怎么用 32
2-15　VC 的编译模式 33
2-16　为什么 Debug 版本程序可以正常运行而 Release 版本无法正常运行 33
2-17　VC 项目文件说明 34
2-18　定制 AppWizard 35

第 3 章　对话框和控件 41

3-1　如何改变对话框的背景颜色 42

3-2 如何为对话框设置一背景图 43
3-3 如何创建和使用非模式对话框 45
3-4 怎样从 MFC 扩展动态链接库(DLL)中显示一个对话框 47
3-5 怎样循环查询对话框上所有控件的 RECT 48
3-6 如何在一个 Dialog 上建立一个简单的超链接 48
3-7 按 ESC 时对话框不退出 49
3-8 如何向基于对话框的应用加上菜单 50
3-9 对话框中如何使用 ToolTip 50
3-10 如何将一个对话框置于最顶层 52
3-11 如何实现对话框的淡入淡出效果 53
3-12 如何创建形状不规则的对话框 55
3-13 实现全屏对话框 61
3-14 实现一个点击不到的按钮 63
3-15 创建位图按钮 66
3-16 创建超链接按钮 68
3-17 创建非矩形的按钮 75
3-18 如何限制编辑框中允许出现的字符 82
3-19 如何向编辑框中追加文本 83
3-20 实现一个自动完成的组合框 84
3-21 在状态条中显示程序运行进度 86

第 4 章 进程和线程 91

4-1 进程的创建和终止 92
4-2 工作线程和用户界面线程 95
4-3 使用事件对象来实现线程的同步 99
4-4 使用临界区对象来实现线程的同步 101
4-5 使用互斥量对象来实现线程的同步 104
4-6 使用信号量对象实现线程的同步 107
4-7 多线程任务调度与处理 110

第 5 章 消息映射与处理 121

5-1 鼠标消息 122
5-2 键盘消息 126
5-3 自定义消息 128
5-4 利用消息来实现进程间通信 129
5-5 利用钩子在多进程中处理 Windows 消息 131
5-6 使用命令范围消息处理函数 135
5-7 重定向消息 136

第 6 章 文件和系统操作 143

6-1 文件夹的选择和拷贝 144
6-2 删除指定路径下的某种类型的文件 148
6-3 当文档(文件)被修改时在标题上给出提示 153
6-4 调用 html 文件的方法 154
6-5 文件映像与内存映射文件 156
6-6 改变系统时间 158
6-7 让系统启动时运行某个应用程序 160
6-8 如何得到光驱的盘符 163
6-9 获得当前程序运行的目录（不包括文件名） 164
6-10 限制软件的使用次数 165
6-11 如何得到 Windows 的版本 167
6-12 得到系统硬件信息 169

第 7 章 窗口和桌面系统 179

7-1 创建不可改变大小和不能移动的窗口 180
7-2 使窗口始终在最前方 180
7-3 在程序运行前禁止窗口右上方的关闭按钮 182
7-4 去除 MFC APPWIZARD 生成工程标题中的“Untitled-MyApp” 183
7-5 恢复窗口位置 184
7-6 改变视的背景颜色 185
7-7 设置 FormView 的背景色 186
7-8 将两个工具条停靠在一行 187
7-9 如何在 View 中创建控件 189
7-10 实现窗口全屏显示和工具栏的飘浮与停靠 192
7-11 在状态栏上添加按钮和组合框 195
7-12 创建标签视 200
7-13 调出开始菜单中的关机对话框 212
7-14 托盘编程 213
7-15 隐藏显示系统任务条 219
7-16 如何得到和改变分辨率 220

第 8 章 图形图像编程 227

8-1 设备坐标、物理坐标和逻辑坐标的区别及相互转换 228
8-2 视口与窗口的区别和转换 229
8-3 CDC、CPaintDC、CClientDC 和 CWindowDC 的区别和应用 229
8-4 各种图形元素的绘制 230
8-5 GDI 对象的访问 231

8-6 使用鼠标绘图 233
8-7 橡皮线的实现 238
8-8 在对话框中绘制图形 240
8-9 使用 MFC 开发 OpenGL 应用程序 244
8-10 利用 OpenGL 实现动画效果 247
8-11 DDB 与 DIB 的区别与相互转换 249
8-12 在用户区显示位图 253
8-13 获取位图的尺寸 255
8-14 显示 256 色位图 256
8-15 对位图进行伸缩和镜像处理 259
8-16 显示 JPG 和 GIF 文件 260

第 9 章 多媒体技术 263

9-1 捕捉窗体内容存储到剪贴板 264
9-2 实现位图动画 266
9-3 打造自己特色的屏幕保护程序 271
9-4 如何播放 Wave、Midi 等文件 276
9-5 如何播放 avi 文件 292

第 10 章 ActiveX 295

10-1 如何快速注册 DLL 和 OCX 文件 296
10-2 Windows 中注册 ActiveX 控件的几种方法 296
10-3 ActiveX 控件制作实例——牌九 ActiveX 控件 299

第 11 章 调试技术 307

11-1 调试环境的建立 308
11-2 调试小述 309
11-3 设置断点 309
11-4 TRACE 宏 312
11-5 ASSERT 宏 313
11-6 VERIFY 宏 313
11-7 在 Console 应用程序中使用 TRACE 宏 313
11-8 内存泄漏的检查 315
11-9 调试 DLL 318
11-10 使用 Dependency Walker 318
11-11 使用 Spy++ 319

第 12 章 VC 数据库编程 321

12-1 如何创建 OBDC 数据源 322

12-2 如何用 ODBC 进行数据库开发 323
12-3 如何动态加载 ODBC 数据源 328
12-4 如何使用 DAO 进行数据库开发 332
12-5 如何对 MFC 工程添加 OLE DB 支持 342
12-6 如何使用 OLE DB 进行数据库编程 342
12-7 如何用#import 指令导入 ADO 类库 348
12-8 如何添加对 ADO 2.0 VC++接口的支持 348
12-9 如何初始化 ADO 环境 348
12-10 如何使用 ADO 进行数据库开发 348
12-11 如何使用 VC++存取数据库中的大对象 354

第 13 章 网络编程 361

13-1 如何从应用程序中打开 URL 362
13-2 如何从应用程序中发送电子邮件 362
13-3 在 Win2000 系统下修改主机名、IP、网关、子网掩码和代理服务器 362
13-4 如何得到多穴主机的多个 IP 地址 374
13-5 如何枚举局域网内的计算机 376
13-6 读取网卡的 Mac 地址 379
13-7 一个小型的公司客服系统——C/S 使用示例 381
13-8 如何在应用程序中映射网络驱动器 389
13-9 如何往 IE 的工具条上添加自定义的图标 391
13-10 利用 WebBrowser 控件创建自己的浏览器 394

第 14 章 帮助系统 397

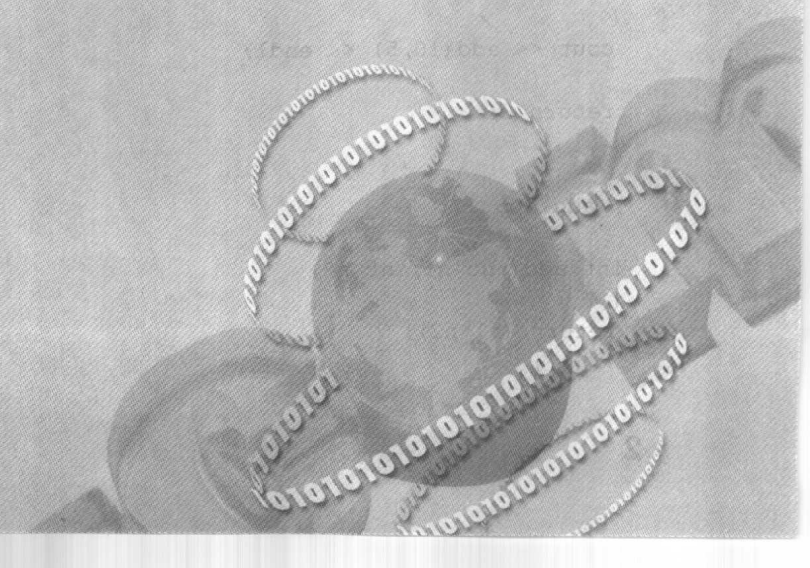
14-1 如何制作一个 hlp 格式的帮助文件 398
14-2 如何制作一个 chm 格式的帮助文件 403
14-3 如何在程序中调用 chm 格式的帮助文件 409

Chapter 1

C/C++基础

本章重点：

扎实的C/C++基础是学好VC的前提。本章针对VC编程中涉及的常见C/C++问题，进行了详尽的分析和论述，同时也给出了相应的代码示例。



1-1 两个类互为成员如何声明

如果 A 类和 B 类互为成员，应该如下声明：

```
class B;

class A
{
 B*  m_b;
};

class B
{
 A   m_a;
     };
```

关键有两点：

1. 其中的一个类需要提前声明（Forward Declaration）。
2. 被提前声明的类在其被具体定义之前只能声明该类的指针或引用，如上面 class A 定义中只能声明 class B 的指针（或引用）。一旦类的定义结束（编译器遇到类定义的右花括号之后），用该类来声明类实例对象或对象指针、引用都是合法的，如上面的类 B 中定义类 A 的实例对象就是正确的，因为在此前类 A 的定义已结束，反之在 A 中声明 B 的对象实例就是错误的。

1-2 inline 函数使用问题

现在有一程序，包括如下两个子程序：

```
//control.cpp

#include "ostream.h"

int add(int n1,int n2);

int main()
{
 cout << add(10,5) << endl;

 return 0;
}

//add.cpp

int add(int n1,int n2)
{
 return (n1+n2);
      }
```

在以上情况下编译正确，但如果将 add.cpp 改成如下形式：

```
//add.cpp

inline int add(int n1,int n2)
{
 return (n1+n2);
      }
```

仅仅增加了一个 inline，则编译报错如图 1-1 所示。

```
Linking...
control.obj : error LNK2001: unresolved external symbol "int __cdecl add(int,int)" (?add@@YAHHH@Z)
Debug/ex1_2_err.exe : fatal error LNK1120: 1 unresolved externals
Error executing link.exe.

ex1_2_err.exe - 2 error(s), 0 warning(s)
```

图 1-1 增加了 inline 声明后的出错信息

注意：上例是 inline 的一个常见的误用，inline 函数是不能 extern 的！

1-3 inline 和 static 有何区别

inline：在调用处不是 call 指令，而是插入函数体，适合于短小的函数，省去参数入栈以及执行 call、ret 指令的过程，提高函数效率。inline 指定函数的存储方式。

static：该函数只能访问类的 static 成员（包括数据和函数），static 数据成员对于该类只有一个 copy，为所有对象共享，在类对象创建前已经可用。static 指定函数操作成员的方式。

1-4 THIS_FILE 表示什么意思

在 VC 生成的框架代码中经常会看到如下的代码片断：

```
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif
```

许多人不明白其中 THIS_FILE 的含义，这里做一个解释：

THIS_FILE 是一个 char 数组全局变量，字符串值为当前文件的全路径，这样在 Debug 版本中当程序出错时出错处理代码可用这个变量告诉你是哪个文件中的代码有问题。

1-5 try 和 TRY 有何区别

try 是 C 新增加的关键字，TRY 是 MFC 定义的宏。

1-6 VC 中的函数调用习惯

进行函数调用时有几种调用方法，主要分为 C 式和 Pascal 式。C 调用方式通过_cdecl 声明，Pascal 调用方式通过 PASCAL 或 CALLBACK 或_stdcall 声明。在 C/C++中缺省采用的是 C 式调用方式，而类成员函数缺省采用 Pascal 式调用方式，二者是有区别的，下面结合实例加以说明。

有如下代码段：

```
//main.cpp

#include "stdio.h"
int add(int& m,int& n);

int main()
{
 int m,n;
 m = 3;
 n = 2;
 add(m,n);

 return 0;
}

int add(int& m,int& n)
{
 return (m+n);
}
```

在 VC 的 Debug Window 中，可以看到各条语句对应的汇编代码如下：

```
1:    //main.cpp
2:
3:    int add(int& m,int& n);
4:
5:    int main()
6:    {
00401020   push        ebp
00401021   mov         ebp,esp
00401023   sub         esp,48h
00401026   push        ebx
00401027   push        esi
00401028   push        edi
00401029   lea         edi,[ebp-48h]
0040102C   mov         ecx,12h
00401031   mov         eax,0CCCCCCCCh
00401036   rep stos    dword ptr [edi]
7:         int m,n;
8:         m = 3;
00401038   mov         dword ptr [ebp-4],3
```

```
9:          n = 2;
0040103F    mov         dword ptr [ebp-8],2
10:         add(m,n);
00401046    lea         eax,[ebp-8]
00401049    push        eax
0040104A    lea         ecx,[ebp-4]
0040104D    push        ecx
0040104E    call        @ILT+5(add) (0040100a)
00401053    add         esp,8
11:
12:         return      0;
00401056    xor         eax,eax
13:     }
00401058    pop         edi
00401059    pop         esi
0040105A    pop         ebx
0040105B    add         esp,48h
0040105E    cmp         ebp,esp
00401060    call        chkesp (004010c0)
00401065    mov         esp,ebp
00401067    pop         ebp
00401068    ret
```

从上面的汇编代码可以看出，在主函数 main 中调用的 add(m,n)（第 10 条语句）被翻译成了如下几条汇编代码：

```
00401046    lea     eax,[ebp-8]             ; 取[ebp-8]地址(ebp-8),存到 eax
00401049    push    eax                     ; 压栈
0040104A    lea     ecx,[ebp-4]             ; 取[ebp-4]地址(ebp-4),存到 ecx
0040104D    push    ecx                     ; 压栈
0040104E    call    @ILT+5(add) (0040100a)  ; 调用 add 函数
00401053    add     esp,8                   ; 恢复栈
```

从以上主函数调用 add 函数的过程可以看出：在调用 add 函数前，首先压栈 ebp-8，然后压栈 ebp-4，之后调用 add 函数，最后通过 esp+8 恢复栈。由此可见，在 C 语言如果使用默认的函数修饰_cdecl（C 调用方式），由主调用函数进行参数压栈并且恢复堆栈。

从下面的代码：

```
00401038    mov         dword ptr [ebp-4],3
0040103F    mov         dword ptr [ebp-8],2
```

还可以发现，ebp-4 中存储的值是 3，而 ebp-8 中存储的值是 2。因此，从上面的压栈过程中还可以看出：在采用 C 调用方式时，函数的参数是从右到左逐个压入堆栈的。

如果对上面的源代码稍作修改，把 add 函数默认的 C 式调用方式改为 Pascal 调用方式，如下所示：

```
//main.cpp
#include "windows.h"

int PASCAL add(int& m,int& n);

int main()
```

```
{
 int m,n;
 m = 3;
 n = 2;
 add(m,n);

 return 0;
}

int PASCAL add(int& m,int& n)
{
 return (m+n);
}
```

则上面的代码对应的汇编代码就变成了：

```
1:       //main.cpp
2:       #include "windows.h"
3:
4:       int PASCAL add(int& m,int& n);
5:
6:       int main()
7:       {
00401020    push        ebp
00401021    mov         ebp,esp
00401023    sub         esp,48h
00401026    push        ebx
00401027    push        esi
00401028    push        edi
00401029    lea         edi,[ebp-48h]
0040102C    mov         ecx,12h
00401031    mov         eax,0CCCCCCCCh
00401036    rep stos    dword ptr [edi]
8:          int m,n;
9:          m = 3;
00401038    mov         dword ptr [ebp-4],3
10:         n = 2;
0040103F    mov         dword ptr [ebp-8],2
11:         add(m,n);
00401046    lea         eax,[ebp-8]
00401049    push        eax
0040104A    lea         ecx,[ebp-4]
0040104D    push        ecx
0040104E    call        @ILT+0(add) (00401005)
12:
13:         return 0;
00401053    xor         eax,eax
14:   }
00401055    pop         edi
00401056    pop         esi
00401057    pop         ebx
```