

**课 程 实 验 报 告**

**课程名称： 汇编语言程序设计实验**

**实验名称： 实验8 Win32程序设计**

**实验时间： 2018-4-28，14：00-17：30 实验地点： 南一楼804室74号实验台**

**指导教师： 左琼 专业班级：计算机科学与技术201603班**

**学 号： U201614577 姓 名： 龙际全**

**同组学生： 无 报告日期： 2018年 5 月 15日**

**原创性声明**

  本人郑重声明：本报告的内容由本人独立完成，有关观点、方法、数据和文献等的引用已经在文中指出。除文中已经注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品或成果，不存在剽窃、抄袭行为。

特此声明！

学生签名：

日期：

成绩评定

|  |  |  |
| --- | --- | --- |
| 实验完成质量得分（70分）（实验步骤清晰详细深入，实验记录真实完整等） | 报告撰写质量得分（30分）（报告规范、完整、通顺、详实等） | 总成绩（100分） |
|  |  |  |

指导教师签字：

                    日期：

**目录**

[1 实验目的与要求 1](#_Toc514159592)

[2 实验内容 1](#_Toc514159593)

[3 实验过程 2](#_Toc514159594)

[3.1 任务1 2](#_Toc514159595)

[3.1.1 环境配置 2](#_Toc514159596)

[3.1.2 源程序 7](#_Toc514159597)

[3.1.3 实验步骤 15](#_Toc514159598)

[3.1.4 实验记录与分析 15](#_Toc514159599)

[4 体会 17](#_Toc514159600)

[5 注 18](#_Toc514159601)

[参考文献 19](#_Toc514159602)

# 实验目的与要求

（1）熟悉WIN32程序的设计和调试方法；

（2）熟悉宏汇编语言中INVOKE、结构变量、简化段定义等功能；

（3）进一步理解机器语言、汇编语言、高级语言之间以及实方式、保护方式之间的一些关系。

# 实验内容

编写一个基于窗口的WIN32程序，实现网店商品信息管理程序的平均利润率计算及商品信息显示的功能（借鉴实验三的一些做法），具体要求如下描述。

功能一：编写一个基于窗口的WIN32程序的菜单框架，具有以下的下拉菜单项：

File Action Help

Exit Average About

List

点菜单File下的Exit选项时结束程序；点菜单Help下的选项About，弹出一个消息框，显示本人信息，类似图5.1所示。点菜单Action下的选项Average、List将分别实现计算平均利润率或显示SHOP1所有商品信息的功能（详见功能二的描述）。



图5.1 菜单示例

功能二：要求采用结构变量存放商品的相关信息。商品数至少定义5种。

点菜单项Average时，按照实验三的方法计算所有商品的平均利润率。用TD32观察计算结果。

点菜单项List时，要求能在窗口中列出SHOP1的所有商品的信息。具体显示格式自行定义，可以参照图5.2的样式（不要求用中文）。



图5.2 商品信息显示示意图

# 实验过程

## 任务1

### 环境配置

实验环境：Microsoft Visual Studio Community 2017下masm32环境。

考虑到masm32中的win32 sdk基本都取自Visual Studio或更低版本的Visual C++，因此从理论上来说选择Visual Studio作为编程环境是完全可行的。具体配置如下：

1. 新建一个项目，项目模板选择Visual C++的空项目，如下图3.1.1，不选择其他项目是因为考虑到Visual Studio在处理C/C++文件和汇编文件时会有所不同。

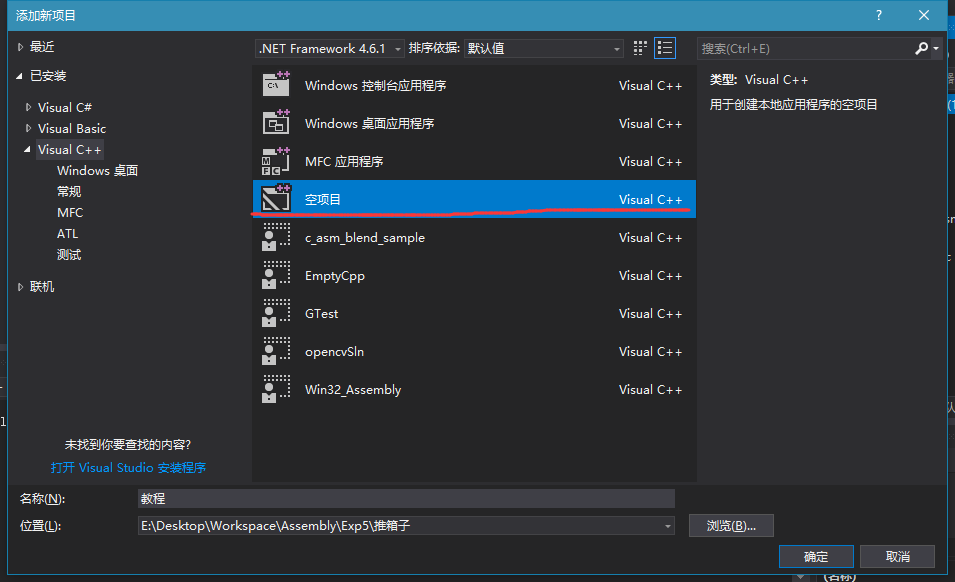


图3.1.1 新建一个空项目

1. 右键该项目，依次选择生成依赖项->生成自定义，如下图3.1.2，选择masm依赖项，如下图3.1.3。

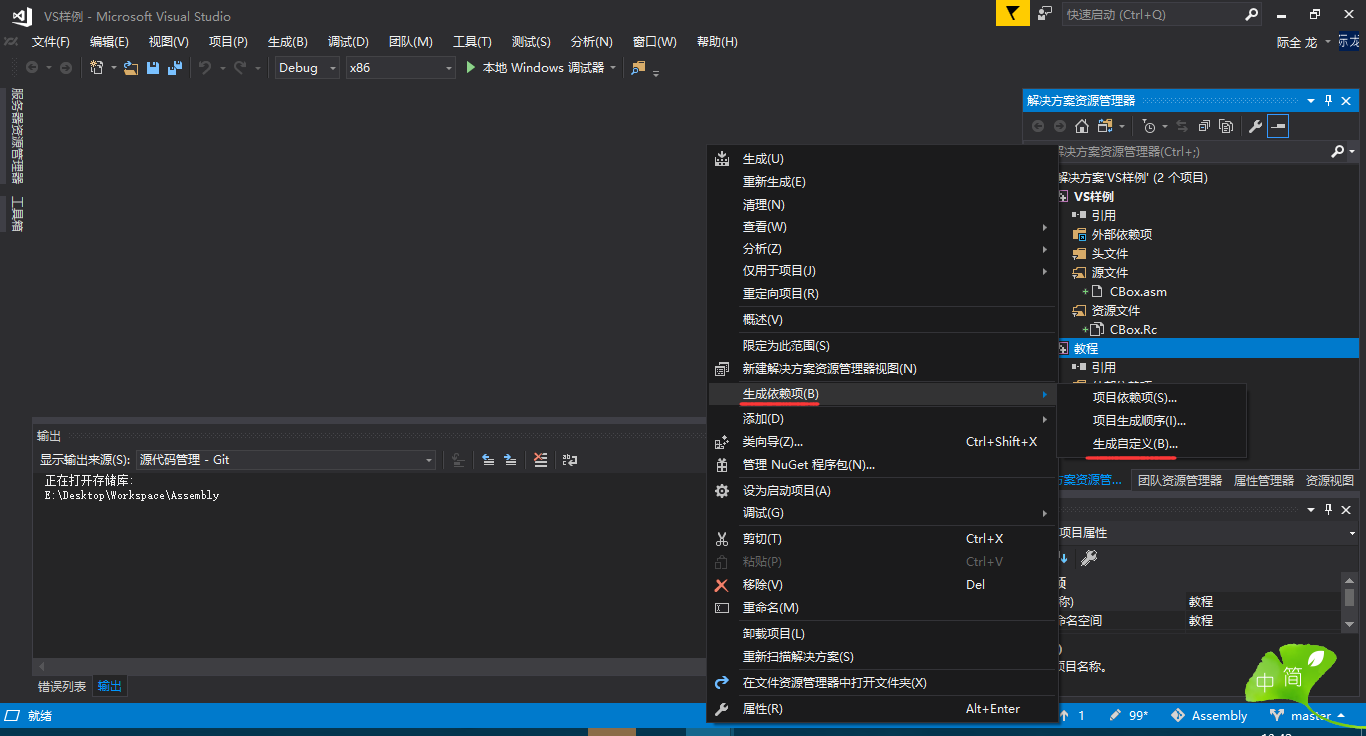


图3.1.2 生成自定义

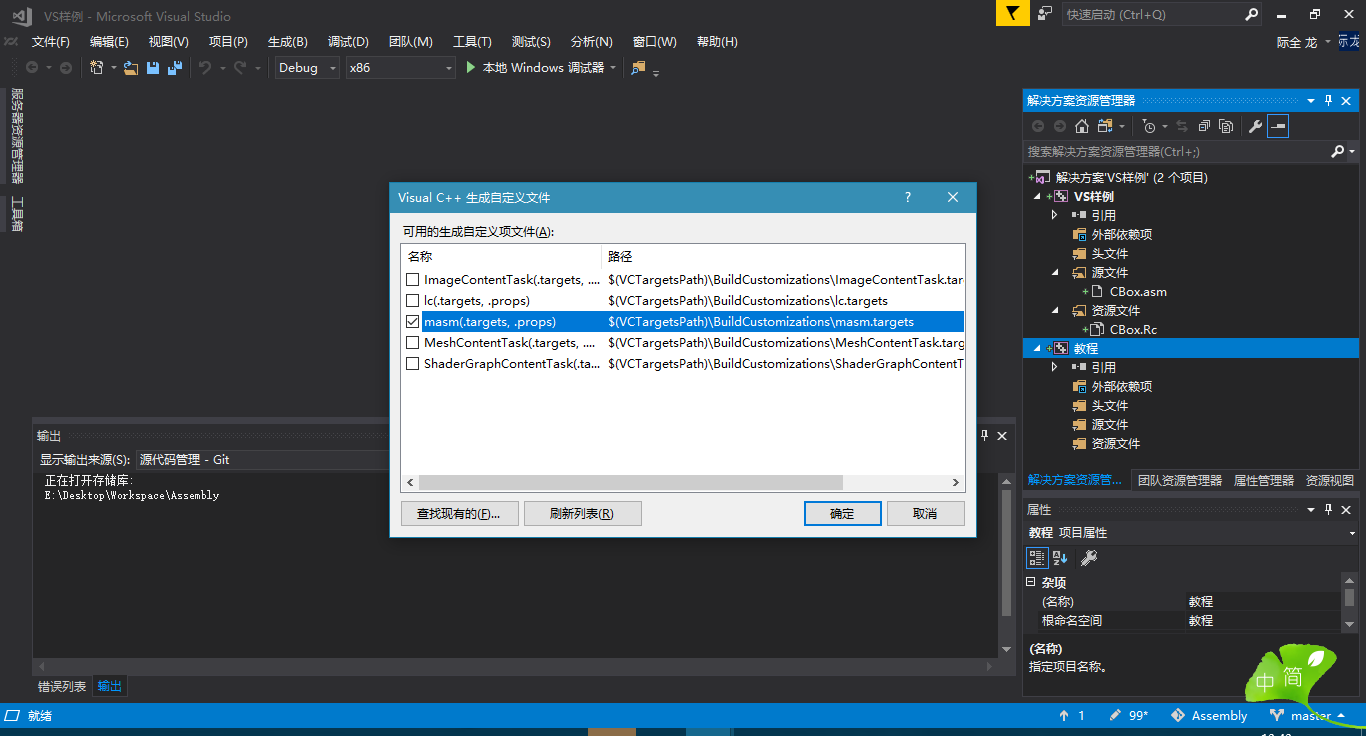


图3.1.3 masm依赖项

1. 右键该项目，选择属性，在属性页中找到 连接器->系统->子系统，选择窗口子系统即/subsystem:windows，如下图3.1.4。

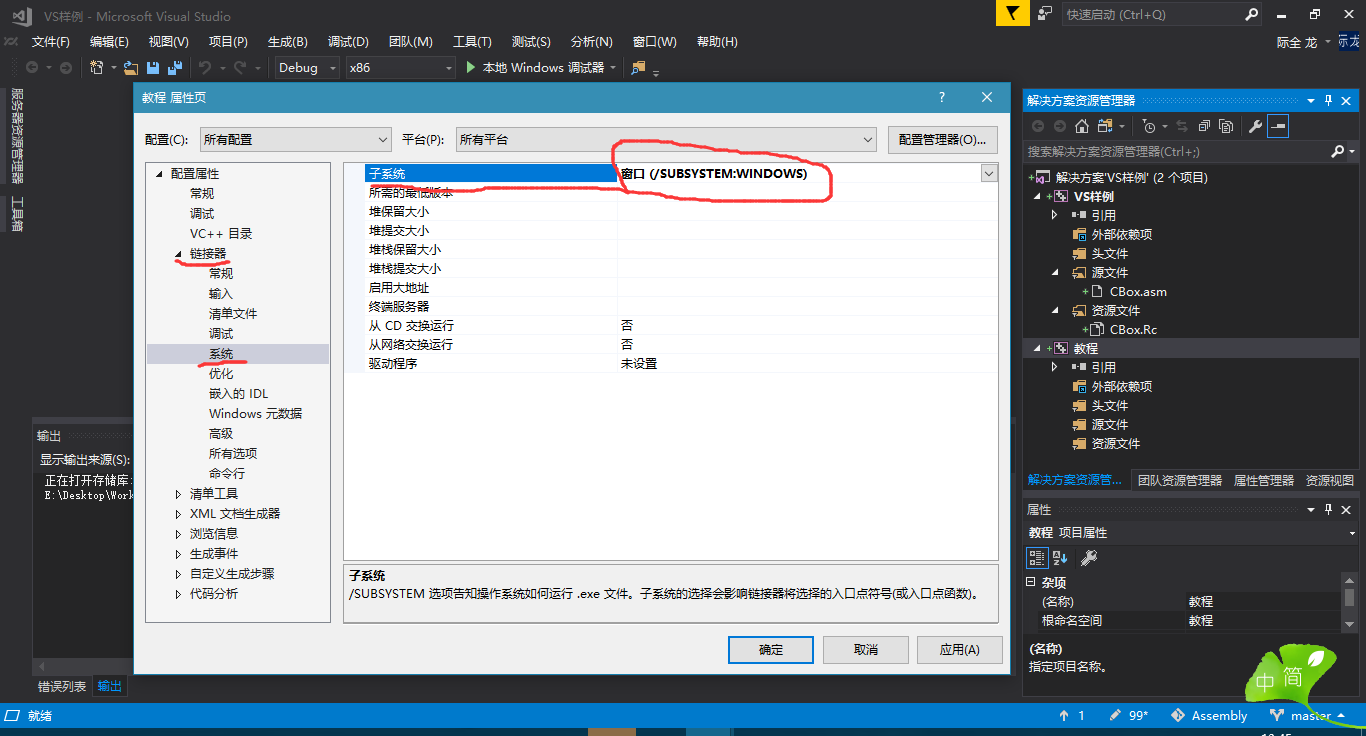


图3.1.4 选择窗口子系统

1. 添加库文件头文件及其他依赖到项目，同理打开属性页，选择 配置属性->VC++目录，包含目录添加masm32 sdk中的include目录，库目录添加masm32中的lib文件夹，Windows运行库目录添加masm32中的dll文件夹。其实按理来说Visual Studio本身就包含这些依赖，因为masm32 sdk就是从中抽取出来的，但是我们为了以防万一还是选择添加这些依赖。如下图3.1.5。

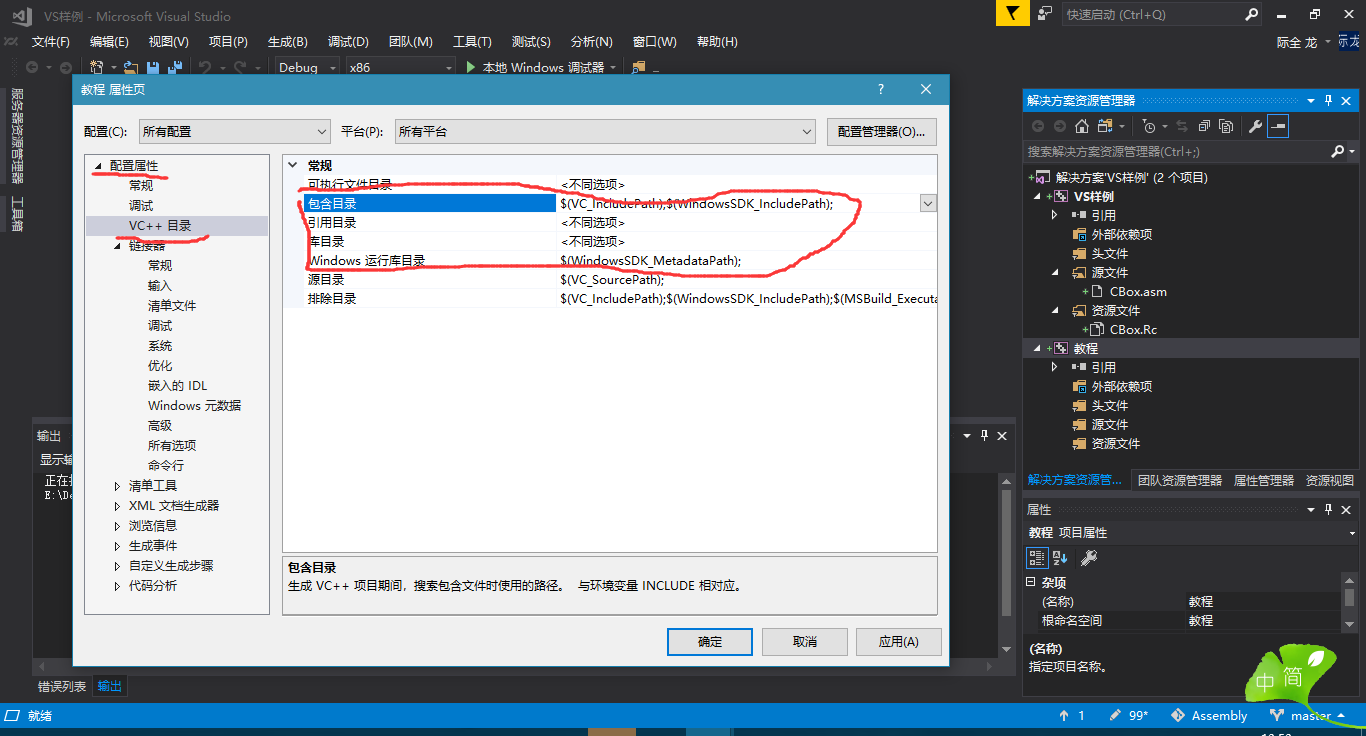


图3.1.5 添加依赖

1. 测试，添加cbox.asm, cbox.rc到项目中，执行该项目，成功了！我们熟悉的推箱子出现在了我们的眼前，如下如3.1.6。

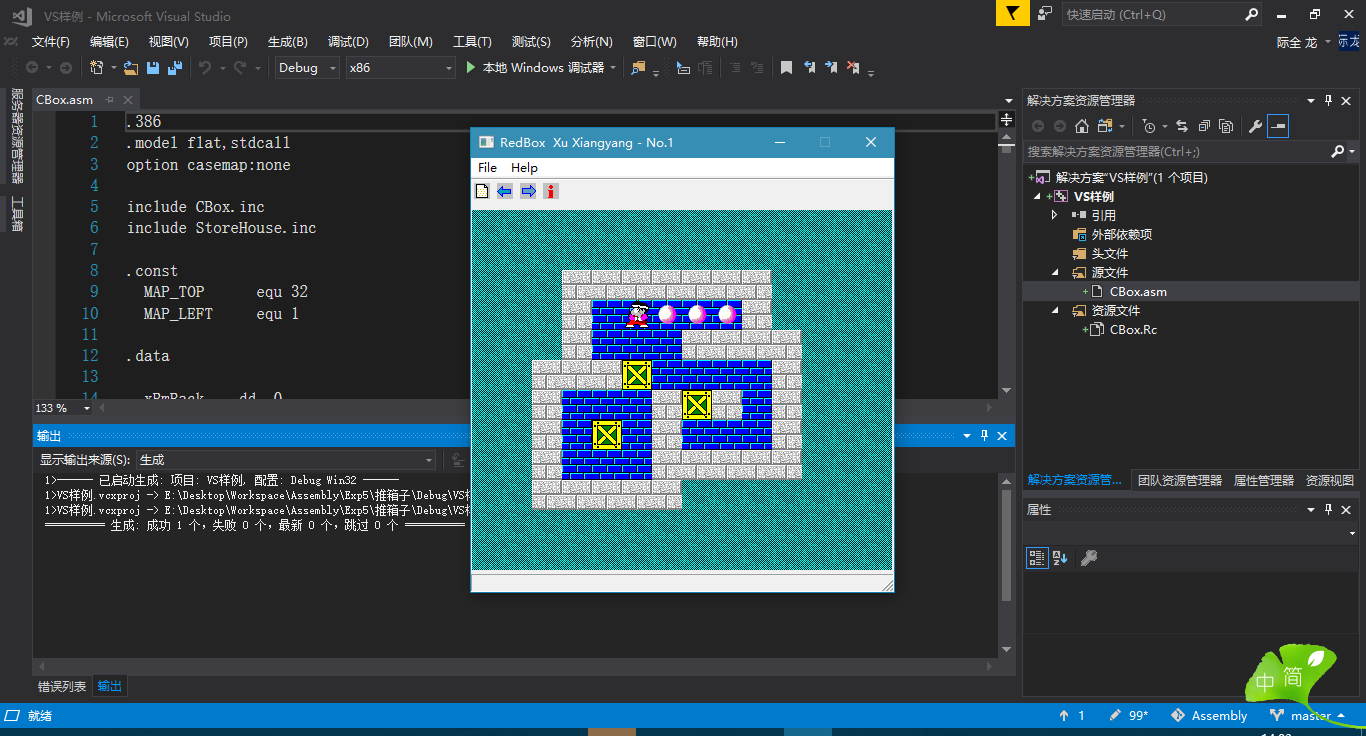


图3.1.6 经典Win32编程案例推箱子

1. 强大的调试功能，Visual Studio作为微软最强大的IDE在调试上当然不输任何人，即使是汇编也不例外。源码级的调试加上友好的内存和寄存器窗口以及优秀的反汇编功能都让我们的工作变得简单。

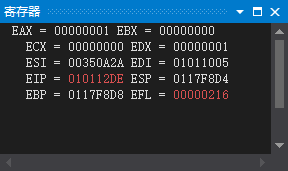


图3.1.7 寄存器窗口，寄存器值改变时变红

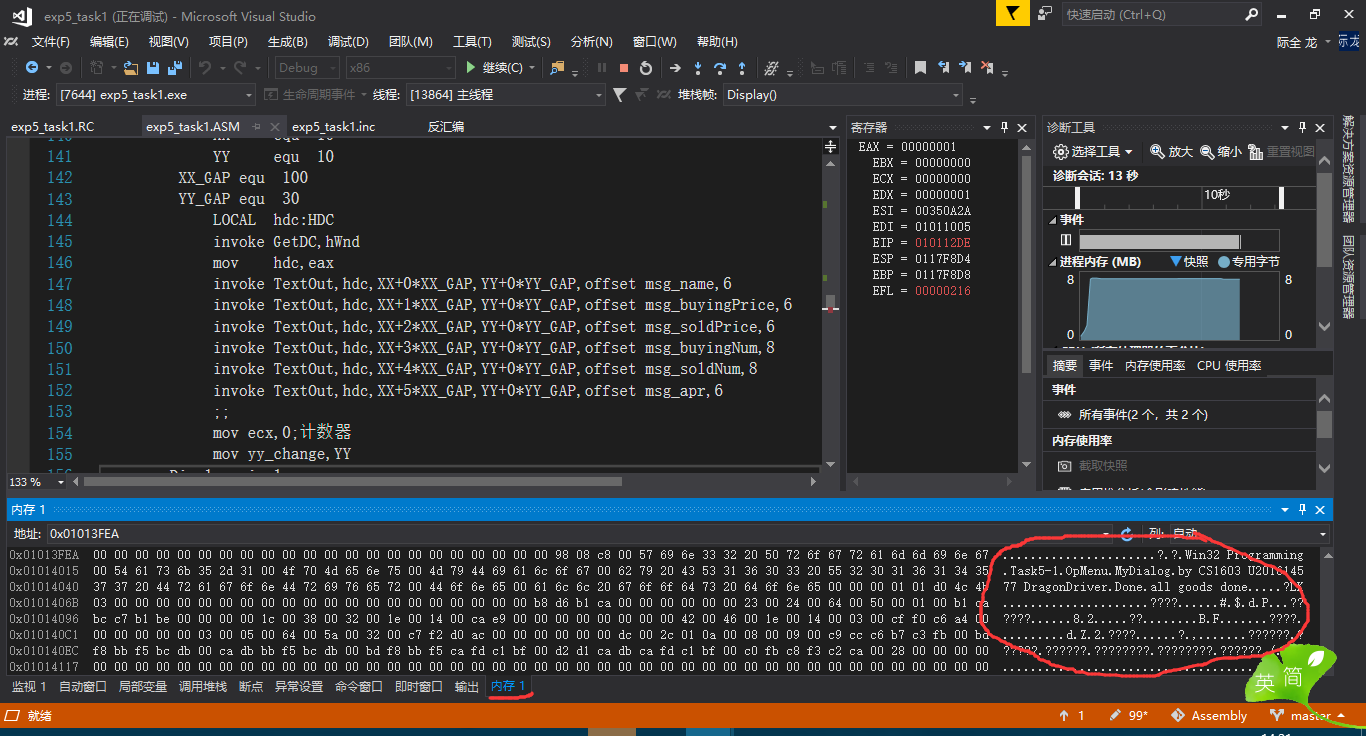


图3.1.8 内存窗口，实时显示内存数据

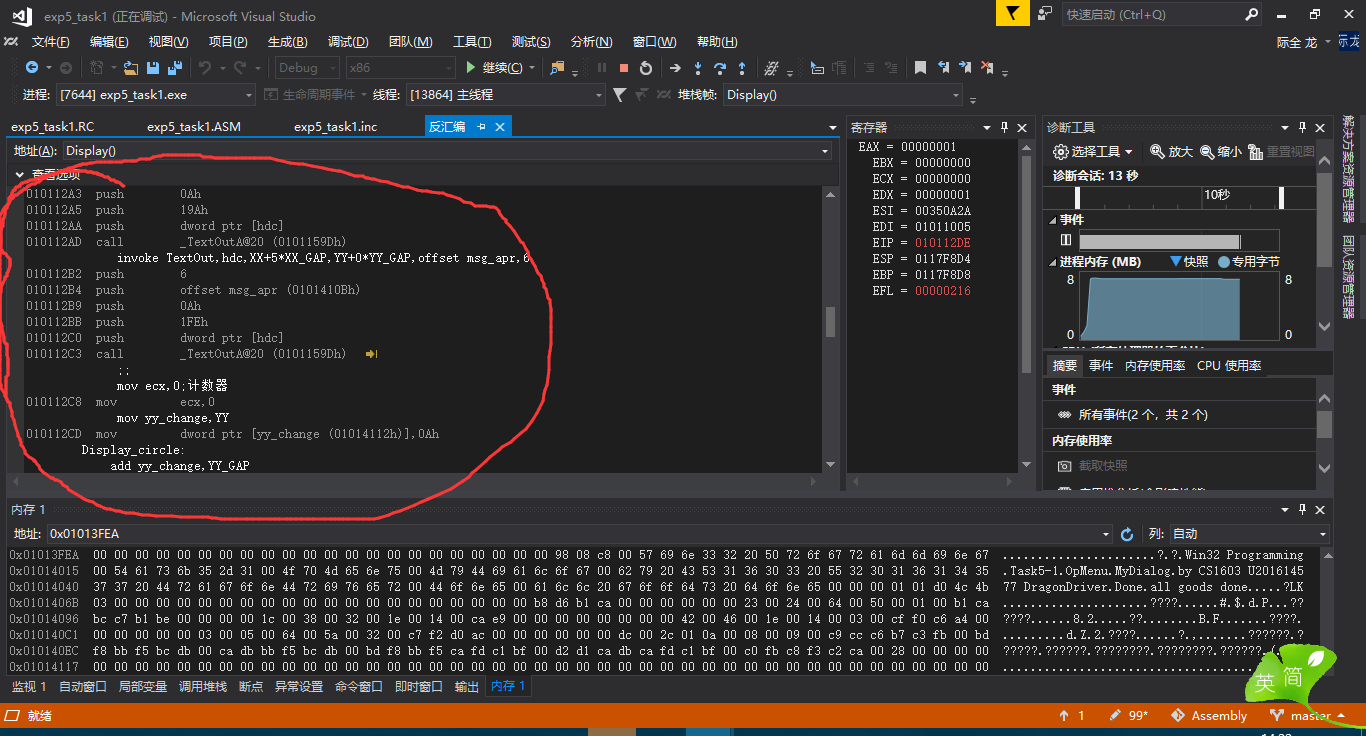


图3.1.9 程序反汇编

1. 配置项目模板，我们希望我们下次在重新使用Visual Studio编写开发汇编语言程序时，不需要想第一次这样去配环境，那我们可以保存项目模板，该项目模板保存了项目的配置信息，我们下次使用该模板可以自动加载配置。Visual Studio项目模板配置详见msdn文档<https://msdn.microsoft.com/zh-cn/library/ms247121.aspx?f=255&MSPPError=-2147217396>。
2. 恭喜我们完成了阶段性的第一步，让我们先稍稍庆祝下我们有一个这么良好的开端以及这么好的编程环境。

### 源程序

1.程序名exp5\_task1.asm

.386

.model flat,stdcall

option casemap:none

WinMain proto :DWORD,:DWORD,:DWORD,:DWORD

WndProc proto :DWORD,:DWORD,:DWORD,:DWORD

Display proto :DWORD

include exp5\_task1.inc

include windows.inc

include user32.inc

include kernel32.inc

include gdi32.inc

include shell32.inc

includelib user32.lib

includelib kernel32.lib

includelib gdi32.lib

includelib shell32.lib

goods struct

goodsName db 10 dup(0)

buyingPrice dw 0

soldPrice dw 0

buyingNum dw 0

soldNum dw 0

apr dw 0

goods ends

.data

cost1 dw 0

profit1 dw 0

goodsNum equ 30

curNum equ 5

ClassName db 'Win32 Programming',0

AppName db 'Task5-1',0

MenuName db 'OpMenu',0

DlgName db 'MyDialog',0

AboutMsg db 'by CS1603 U201614577 DragonDriver',0

AverageMsg db 'Done',0

AverageReminder db 'all goods done',0

hInstance dd 0

CommandLine dd 0

numString db 20 dup(0)

shop goods <'钢笔',35,36,100,80,0>

goods <'笔记本',28,56,50,30,0>

goods <'书',66,70,30,20,0>

goods <'橡皮',3,5,100,90,0>

goods <'球鞋',220,300,10,8,0>

msg\_name db '商品名',0

msg\_buyingPrice db '进货价',0

msg\_soldPrice db '售货价',0

msg\_buyingNum db '进货数量',0

msg\_soldNum db '已售数量',0

msg\_apr db '利润率',0

yy\_change dd 0

xx\_change dd 0

strLength dd 0

.code

Start: invoke GetModuleHandle,NULL

mov hInstance,eax

invoke GetCommandLine

mov CommandLine,eax

invoke WinMain,hInstance,NULL,CommandLine,SW\_SHOWDEFAULT

invoke ExitProcess,eax

;;

WinMain proc hInst:DWORD,hPrevInst:DWORD,CmdLine:DWORD,CmdShow:DWORD

LOCAL wc:WNDCLASSEX

LOCAL msg:MSG

LOCAL hWnd:HWND

invoke RtlZeroMemory,addr wc,sizeof wc

mov wc.cbSize,SIZEOF WNDCLASSEX

mov wc.style, CS\_HREDRAW or CS\_VREDRAW

mov wc.lpfnWndProc, offset WndProc

mov wc.cbClsExtra,NULL

mov wc.cbWndExtra,NULL

push hInst

pop wc.hInstance

mov wc.hbrBackground,COLOR\_WINDOW+1

mov wc.lpszMenuName, offset MenuName

mov wc.lpszClassName,offset ClassName

invoke LoadIcon,NULL,IDI\_APPLICATION

mov wc.hIcon,eax

mov wc.hIconSm,0

invoke LoadCursor,NULL,IDC\_ARROW

mov wc.hCursor,eax

invoke RegisterClassEx, addr wc

INVOKE CreateWindowEx,NULL,addr ClassName,addr AppName,\

WS\_OVERLAPPEDWINDOW,CW\_USEDEFAULT,\

CW\_USEDEFAULT,CW\_USEDEFAULT,CW\_USEDEFAULT,NULL,NULL,\

hInst,NULL

mov hWnd,eax

INVOKE ShowWindow,hWnd,SW\_SHOWNORMAL

INVOKE UpdateWindow,hWnd

;;

MsgLoop: INVOKE GetMessage,addr msg,NULL,0,0

cmp EAX,0

je ExitLoop

INVOKE TranslateMessage,addr msg

INVOKE DispatchMessage,addr msg

jmp MsgLoop

ExitLoop: mov eax,msg.wParam

ret

WinMain endp

WndProc proc hWnd:DWORD,uMsg:DWORD,wParam:DWORD,lParam:DWORD

LOCAL hdc:HDC

.IF uMsg == WM\_DESTROY

invoke PostQuitMessage,NULL

.ELSEIF uMsg == WM\_KEYDOWN

.IF wParam == VK\_F1

;;your code

.ENDIF

.ELSEIF uMsg == WM\_COMMAND

.IF wParam == IDM\_FILE\_EXIT

invoke SendMessage,hWnd,WM\_CLOSE,0,0

.ELSEIF wParam == IDM\_ACTION\_LIST

call Average

invoke Display,hWnd

.ELSEIF wParam == IDM\_HELP\_ABOUT

invoke MessageBox,hWnd,addr AboutMsg,addr AppName,0

.elseif wParam== IDM\_ACTION\_AVERAGE

call Average

invoke MessageBox,hWnd,addr AverageReminder,addr AverageMsg,0

.ENDIF

;.ELSEIF uMsg == WM\_PAINT

;;redraw window again

.ELSE

invoke DefWindowProc,hWnd,uMsg,wParam,lParam

ret

.ENDIF

xor eax,eax

ret

WndProc endp

Display proc hWnd:DWORD

XX equ 10

YY equ 10

XX\_GAP equ 100

YY\_GAP equ 30

LOCAL hdc:HDC

invoke GetDC,hWnd

mov hdc,eax

invoke TextOut,hdc,XX+0\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_name,6

invoke TextOut,hdc,XX+1\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_buyingPrice,6

invoke TextOut,hdc,XX+2\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_soldPrice,6

invoke TextOut,hdc,XX+3\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_buyingNum,8

invoke TextOut,hdc,XX+4\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_soldNum,8

invoke TextOut,hdc,XX+5\*XX\_GAP,YY+0\*YY\_GAP,offset msg\_apr,6

;;

mov ecx,0;计数器

mov yy\_change,YY

Display\_circle:

add yy\_change,YY\_GAP

;mov yy\_change,YY+(ecx+1)\*YY\_GAP

mov edx,offset shop

add edx,ecx

pushad

mov esi,edx

call stringLength

invoke TextOut,hdc,XX+0\*XX\_GAP,yy\_change,edx,eax

popad

mov edi,10

mov xx\_change,XX

Display\_item\_circle:

mov ax,word ptr shop[ecx+edi]

movsx eax,ax

lea esi,numString

call SetStringZero

mov ebx,10;十进制

call radix

pushad

lea esi,numString

call stringLength

mov strLength,eax

add xx\_change,XX\_GAP

invoke TextOut,hdc,xx\_change,yy\_change,offset numString,strLength

popad

add edi,2

cmp edi,20

jne Display\_item\_circle

add ecx,20

cmp ecx,20\*curNum

jne Display\_circle

Display\_ret:

ret

Display endp

Average proc

push ebx

push eax

push edx

mov ebx,0;计数器

Average\_core:

mov ax,word ptr shop[ebx+10]

imul word ptr shop[ebx+14];ax=cost

mov cost1,ax

mov ax,word ptr shop[ebx+12]

imul word ptr shop[ebx+16]

sub ax,cost1

mov profit1,ax

mov dx,0

mov ax,profit1

imul eax,word ptr 100

cwd

idiv cost1

mov word ptr shop[ebx+18],ax

add ebx,20

cmp ebx,curNum\*20

jne Average\_core

Average\_ret:

pop edx

pop eax

pop ebx

ret

Average endp

;子程序名称：SetStringZero

;功能：将以ds:esi为指针的字符串内容清零

;参数：ds:esi指向字符串首地址，

;返回：无

;注意事项：字符串以0结尾

;作者：boyjqlong@foxmail.com

SetStringZero proc

push edx

push eax

push esi

push ebx

mov ebx,0

SetStringZero\_core:

mov dl,byte ptr [ebx+esi]

cmp dl,0

je SetStringZero\_ret;等于0时跳出循环

mov byte ptr [ebx+esi],0

inc ebx

jmp SetStringZero\_core

SetStringZero\_ret:

pop ebx

pop esi

pop eax

pop edx

ret

SetStringZero endp

;子程序名称：stringLength

;功能：将EAX中的无符号二进制数

;转换为P进制数

;参数：

;esi为字符串首地址

;返回：

;eax=字符串长度

;作者：boyjqlong@foxmail.com

stringLength proc

push ebx

mov eax,0

stringLength\_count:

mov bh,byte ptr [esi]

cmp bh,0

je stringLength\_ret

inc eax

inc esi

jmp stringLength\_count

stringLength\_ret:

pop ebx

ret

stringLength endp

;子程序名称：radix

;功能：将EAX中的无符号二进制数

;转换为P进制数

;参数：

;EAX--存放带转换的无符号二进制数

;EBX--存放要转换数制的基数

;eSI--存放转换后的P进制ASCII码数字串的字节缓冲区首址

;返回：

;所求P进制ASCII码数字串按高位在前、地位在后的顺序存放在以SI为指针的字节缓冲区中

;eSI--指向字节缓冲区中最后一个ASCII码的下一个字符处

;作者：来自华工80x86汇编语言程序设计课本136页

radix proc

push cx

push edx

xor cx,cx

radix\_lop1:

xor edx,edx

div ebx

push dx

inc cx

or eax,eax

jnz radix\_lop1

radix\_lop2:

pop ax

cmp al,10

jb radix\_l1

add al,7

radix\_l1:

add al,30H

mov [esi],al

inc esi

loop radix\_lop2

pop edx

pop CX

ret

radix endp

end Start

2.头文件exp5\_task1.inc

IDM\_FILE\_EXIT EQU 10001

IDM\_ACTION\_AVERAGE EQU 10101

IDM\_ACTION\_LIST EQU 10102

IDM\_HELP\_ABOUT EQU 10201

3.菜单资源文件exp5\_task1.rc

#define IDM\_FILE\_EXIT 10001

#define IDM\_ACTION\_AVERAGE 10101

#define IDM\_ACTION\_LIST 10102

#define IDM\_HELP\_ABOUT 10201

OpMenu MENU

BEGIN

POPUP "&File"

BEGIN

MENUITEM "E&xit", IDM\_FILE\_EXIT

END

POPUP "Action"

BEGIN

MENUITEM "Average",IDM\_ACTION\_AVERAGE

MENUITEM "L&ist", IDM\_ACTION\_LIST

END

POPUP "&Help"

BEGIN

MENUITEM "&About", IDM\_HELP\_ABOUT

END

END

### 实验步骤

1.根据给出的菜单框架进行修改，将功能区填入自己的代码。

2.编写主要功能模块Average，该模块用于计算商品的利润率，采用循环实现。

3.编写主要功能模块List，该模块用于显示所有商品的信息，采用双重循环实现，第一重循环控制商品，第二重循环控制商品的各种属性。

4.用Visual Studio调试知道功能完善即可。

### 实验记录与分析

1.实验环境条件：AMD A8-8600P 1.6GHz，8G内存；WINDOWS 10下Microsoft Visual Studio Community 2017。

2.编译工程，无错误产生，如下图3.4.1所示：

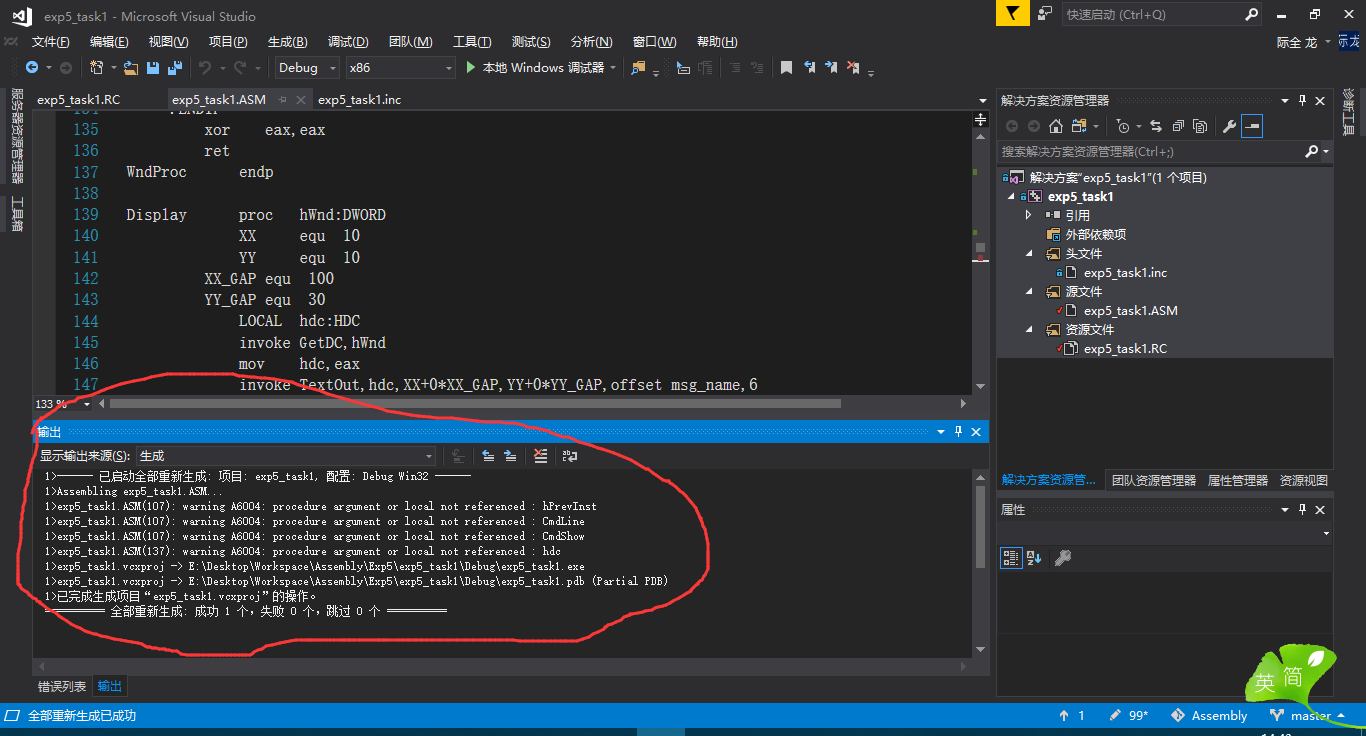


图3.4.1 编译成功

3.点击Average计算商品利润率，计算完成弹出对话框，如下图3.4.2所示；

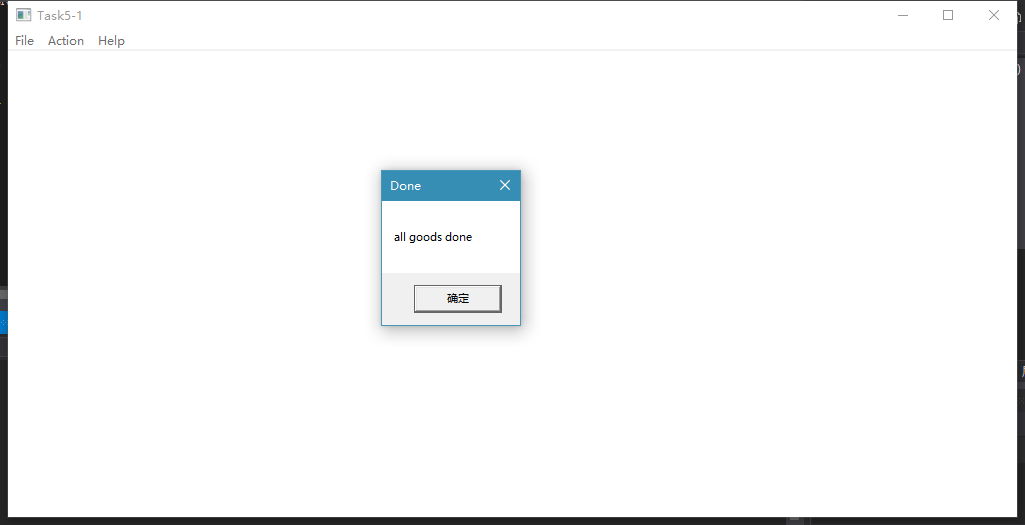


图3.4.2 Average计算利润率

4.点击List，输出所有商品信息，如下图3.4.3：

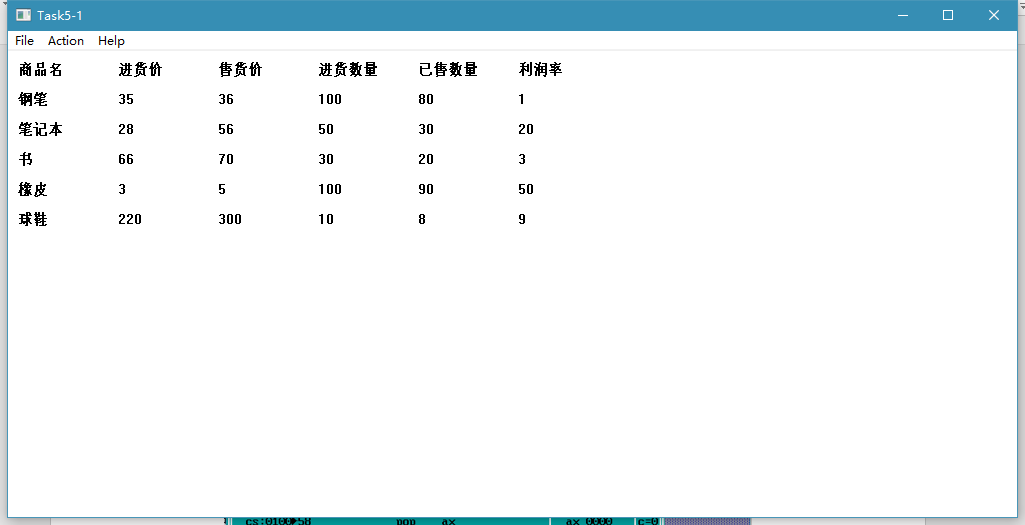


图3.4.3 List输出所有商品信息

5.点击About，弹出软件作者,如下图3.4.4所示：

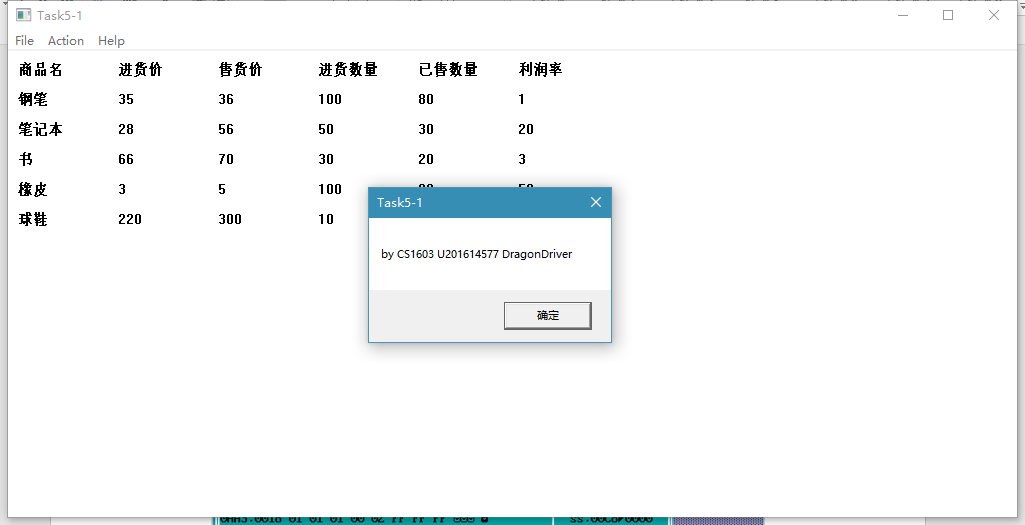


图3.4.4 About信息

# 体会

这次实验让我获益良多，具体为：

1. 锻炼了自己配置编程环境的能力。
2. 提升了编程效率。
3. 增加了对自己解决问题能力的信心。

# 注

有个别实验代码较多较长，不适于直接在word中直接操作。因此本人将本次实验所有代码及二进制可执行文件上传至GitHub，具体网址为<https://github.com/DragonDriver/AssemblyExp/tree/master/Exp5>。

# 参考文献

[1] 王元珍 曹忠升 韩宗芬.《80x86汇编语言程序设计》.华中科技大学出版社