

北京理工大学

本科生课程设计

汇编语言与接口技术个人实验报告

Personal experiment report on assembly language and interface
technology

学 院:	计算机学院
专 业:	计算机科学与技术
学生姓名:	张驰
学 号:	1120191600
指导教师:	李元章

2022 年 4 月 15 日

原创性声明

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名： 张驰 日期： 2022 年 4 月 15 日

关于使用授权的声明

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名： 张驰 日期： 2022 年 4 月 15 日

指导老师签名： 日期： 2022 年 4 月 15 日

汇编语言与接口技术个人实验报告

摘 要

本文使用汇编语言，以 visual studio 2019 作为开发的 IDE，进行了个人实验编程，其中包含大数相乘，浮点数运算计算器，两个文本文件内容的比对。

从中掌握了基本的汇编语言技术，进一步加深了对汇编语言的理解。

关键词：汇编语言与接口技术；大数相乘；浮点运算计算机；文件对比

Personal experiment report on assembly language and interface technology

Abstract

This paper uses assembly language and visual studio 2019 as the IDE to carry out personal experimental programming, including large number multiplication, floating point calculator and the comparison of the contents of two text files.

Master the basic assembly language technology, and further deepen the understanding of assembly language.

Key Words: Assembly language and interface technology; Multiplication of large numbers; Floating point computer; Document comparison

目 录

摘 要	I
Abstract.....	II
第 1 章 实验目的	1
第 2 章 实验环境	1
第 3 章 实验内容 1：大数相乘.....	1
3.1 实验目的	1
3.2 程序设计的基本流程	1
3.3 程序各个模块的具体实现	1
3.3.1 计算字符串长度 strlen 模块	1
3.3.2 字符串处理函数（将字符串转化为逆序数组）	2
3.3.3 大整数乘法模块（核心模块）	3
3.3.4 进位模块	4
3.3.5 输出模块	5
3.3.5 输出模块	6
3.4 程序运行结果	7
第 4 章 实验内容 2：Windows 界面浮点数编程.....	9
4.1 实验目的	9
4.2 界面设计与界面编程代码展示：	9
4.3 程序整体思路：	13
4.4 程序中各个功能模块的介绍：	13
4.5 程序运行结果	17
第 5 章 实验内容 3：文本文件内容对比.....	20
5.1 实验目的	20
5.2 程序设计的基本流程以及界面设计	20
5.3 程序核心模块的介绍	22
5.3.1 文件对比模块 CompareFile:.....	22
5.3.2 读取一行模块 ReadLine:.....	24
5.4 程序运行结果	24
第 6 章 实验心得体会	26
致 谢	27

第 1 章 实验目的

本次实验通过使用ASM汇编语言编写三个程序，从而掌握Windows系统汇编程序的基本结构，掌握基本的汇编指令。通过本次实验，熟练进行汇编程序的编写和调试，学会调用C库函数，提高个人的汇编编程能力，加强对汇编指令的理解。

第 2 章 实验环境

本次实验在Windows10系统下进行，使用MASM, 以Visual Studio 2019作为开发的IDE。

第 3 章 实验内容 1：大数相乘

3.1 实验目的

大数相乘，要求实现两个十进制大整数的相乘（100位以上），输出乘法运算的结果。

3.2 程序设计的基本流程

程序设计的流程如下所示：

使用C语言的scanf函数读取两个数字型的字符串，之后将两个字符串的数字形式转化为逆序数组存放的形式，将两个数组按位相乘并放到结果数组中，对运算后的结果数组进行进位处理。处理完成后，先计算符号位的输出结果，计算‘-’的个数后判断是否输出‘-’，之后输出运算后的结果。

3.3 程序各个模块的具体实现

该部分就整个汇编程序的各个模块的具体实现进行介绍。

3.3.1 计算字符串长度 strlen 模块

代码如下所示：

```
1. strlen proc stdcall USES edi string : ptr byte
2.  mov ax, SEG strlen
3.  mov edi, string
4.  mov eax, 0
5.  strlen_1:
6.  CMP byte ptr [edi], 0; 判断是否等于 0, 即\0
7.  JZ strlen_2
8.  ADD edi, 1
9.  ADD eax, 1
10. jmp strlen_1
11. strlen_2:
12. RET
13. strlen ENDP
```

逐个字符遍历字符串，直到遍历至 ‘\0’ 字符，在遍历字符串时，逐个递增寄存器EAX的内容，之后返回，EAX的内容即为该字符串的长度。

其中 USES 后使用的寄存器表示会自动保存并恢复寄存器的值，该函数的返回值保存到寄存器EAX中。

3.3.2 字符串处理函数（将字符串转化为逆序数组）

本部分代码如下所示：

```
1. ProcessString proc stdcall USES edx ecx eax string: ptr BYTE, numbers: ptr DWORD, len:DWORD
2.  XOR  ah,ah
3.
4.  MOV  edx, string
5.  MOV  ebx, numbers
6.  MOV  ecx, len; //判断长度，逐步递减
7.  SUB  ecx, 1
8.
9.  JMP  PS_2
10.
11. PS_1:
12. MOV  AL, [edx][ecx]
13. CMP  AL, 2DH
14. JZ   PS_3; 判断 AL 是否为 ‘-’，是 ‘-’ 时就跳转到 PS_3
15. SUB  AL, 30H
16.
17. MOV  [EBX], EAX
18. ADD  EBX, 4
```

```

19. SUB ECX, 1
20.
21. PS_2:
22. CMP ecx, 0
23. JNS PS_1
24. RET
25.
26. PS_3:
27. ADD negflag, 1
28. RET
29.
30. ProcessString ENDP

```

遍历字符串，将字符串的每个字符减去字符‘0’，之后将该数逆序放置到数组中，数组存放在EBX寄存器中。

其中，需要注意的是对‘-’的处理。这里采用Negflag 的方式做标记，当遇到‘-’时，将全局变量negflag自增。在最后处理时，如果negflag == 1，那么就输出‘-’表示结果是负数，其余情况下均为正数。

3.3.3 大整数乘法模块（核心模块）

本部分为核心模块，C语言代码如下所示：

```

1. for (int i = 0; i < len1; i++) {
2.     for (int j = 0; j < len2; j++) {
3.         res[i + j] += a[i] * b[j];
4.     }
5. }

```

本部分汇编代码如下所示：

```

1. Mult_num proc stdcall uses ecx edx ebx edi esi numbers1: ptr dword, length1: dword, numbers2: ptr dword, length2: dword, res: ptr dword
2.     MOV EDX, numbers1
3.     MOV EBX, numbers2
4.     MOV EAX, res
5.     MOV esi, 0
6.     JMP MUL_4
7.
8. MUL_1:
9.     MOV EDI, 0; 第一层循环的主体
10.    JMP MUL_3

```



```
11.
12. MUL_2: ;第二层循环的主体
13. MOV ECX, [EDX + ESI * 4]
14. IMUL ecx, [ebx + edi* 4] ;num1[i]*num2[j]
15.
16. add edi, esi
17. add [eax + edi * 4], ecx ;res[i+j] += num1[i]*num2[j]
18. sub edi, esi
19. inc edi
20.
21. MUL_3: ;第二层循环的判断条件
22. CMP EDI, length2
23. JL MUL_2
24. INC esi
25.
26. MUL_4: ;第一层循环的判断条件
27. CMP esi, length1
28. JL MUL_1
29.
30. RET
31. Mult_num ENDP
```

在程序代码中，包含有两层循环，首先跳转到第一层判断，第一层判断的条件放在ESI变址寄存器中，判断ESI是否小于len1，如果小于len1，跳转到第一层的主体，第一层的主体只有初始化EDI为0，然后进入第二层的判断，判断EDI是否小于len2，满足则跳转到第二层主体。

第二层主体的内容为计算num1[i]*num2[j]，实现res[i+j] += num1[i] * num2[j]。其中，res的结果存储到EAX寄存器中，作为该模块的输出部分。

3.3.4 进位模块

进位模块的汇编代码如下所示：

```
1. carry proc stdcall USES ebx edi edx res:ptr dword
2. local Len
3.
4. invoke numlen, res
5. mov Len, eax
6.
7. mov ebx, res
8. mov edi, 0
```

```
9. C1:
10.    cmp edi, Len
11.    jnl C2
12.
13.    mov eax, [ebx + edi * 4]
14.    xor edx, edx
15.    mov ecx, 10
16.    div ecx
17.    ; eax 中存放除法结果 ,   edx 中存放除法余数
18.    mov [ebx + edi * 4], edx
19.    add [ebx + edi * 4 + 4], eax
20.
21.    inc edi
22.    jmp C1
23. C2:
24.    invoke numlen, res
25.    ret
26. carry endp
```

进位部分非常简单，顺序扫描结果数组，对该数做除法运算，将商进位到下一位，余数作为这一位。

在这一模块中，使用了local来定义该函数中的局部变量，定义Len的局部变量，表示该结果的长度。

3.3.5 输出模块

输出模块使用了printf函数，需要提前进行定义：

```
1. printf PROTO C : ptr sbyte, :VARARG

1. print PROC stdcall uses ebx edi edx ans:ptr dword, len: dword
2.    mov ebx, ans
3.    mov edi, len
4.    sub edi, 1
5. P1:
6.    cmp EDI, 0
7.    JNGE P2; EDI < 0
8.    invoke printf, offset printmsg_d, dword ptr [ebx + edi*4]
9.    SUB edi, 1
10.   jmp P1
11. P2:
12.   mov len1, eax
```

```
13. INVOKE printf, offset huanhang
14. RET
15. print endp
```

整体逻辑比较简单，即进行倒序输出，当EDI寄存器的值等于0时，跳出函数。

其中，输出printf函数，调用的printfmsg_d为“%d”，数字为[ebx+edi*4]，为输出printf(“%d”, [ebx + edi*4]);

3.3.5 输出模块

程序的主模块代码如下所示：

```
1. main PROC
2. INVOKE printf, offset input
3. INVOKE scanf, offset scanfmsg_s, offset string1
4. INVOKE scanf, offset scanfmsg_s, offset string2
5.
6. INVOKE strlen, offset string1;计算数 1 的长度
7. mov len1, eax
8.
9. INVOKE strlen, offset string2;计算数 2 的长度
10. mov len2, eax
11.
12. ;对两个数进行处理
13. INVOKE ProcessString, offset string1, offset nums1, len1
14. INVOKE ProcessString, offset string2, offset nums2, len2
15.
16. INVOKE Mult_num, OFFSET nums1, len1, offset nums2, len2, offset
    numans ;大数乘法
17. INVOKE carry, offset numans ;进行进位
18. MOV lennum, eax ;计算长度
19.
20. MOV edi, negflag
21. CMP edi, 1 ;判断，如果不等于 1 就跳过
22. JNZ M1
23.
24. invoke printf, offset neg_flag
25.
26. M1:
27. INVOKE print, offset numans, lennum ;输出结果
28.
29. INVOKE printf, offset endoutput
```

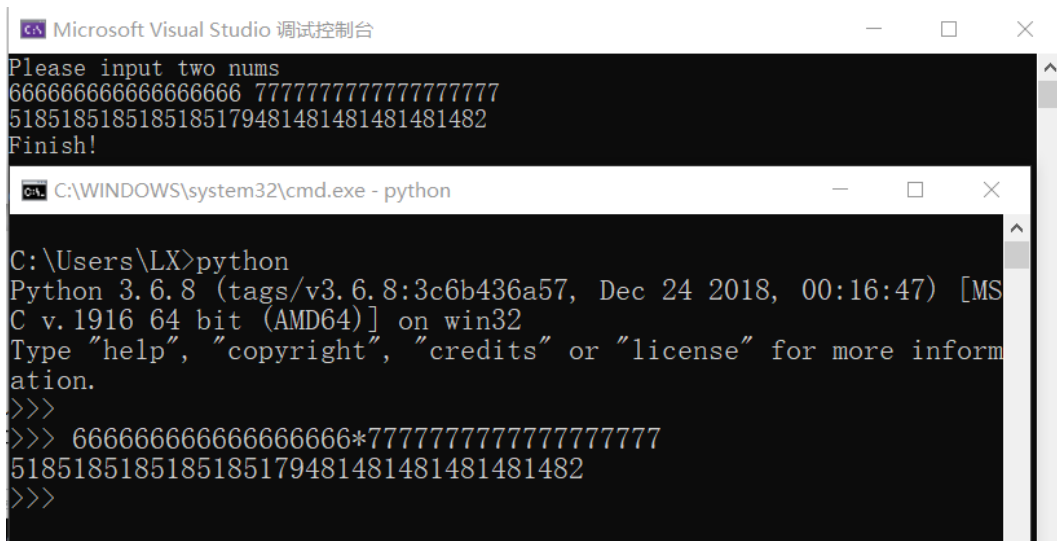
```
30. RET
31.main endp
```

主程序的模块即按照以下流程调用函数进行处理：

1. 使用scanf函数读入两个大整数，以字符串的形式进行存放
2. 调用strlen函数计算两个字符串的长度
3. 将两个数字字符串转换到逆序数组进行存放
4. 根据大整数乘法进行计算，放入到结果数组中
5. 进行进位处理
6. 判断是否输出 ‘-’
7. 输出结果

3.4 程序运行结果

程序运行结果如下所示，将程序运行结果与python运行结果进行比较，完全正确，其中包含有大正整数相乘，两个负数相乘以及正数与负数相乘。



The image shows two windows side-by-side. The top window is the 'Microsoft Visual Studio 调试控制台' (Debug Console). It contains the following text: 'Please input two nums', followed by two lines of input: '66666666666666666666 77777777777777777777' and '5185185185185185179481481481481481482', and finally 'Finish!'. The bottom window is a Windows command prompt titled 'C:\WINDOWS\system32\cmd.exe - python'. It shows the command 'python' being executed, followed by the Python version and architecture information: 'Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MS C v.1916 64 bit (AMD64)] on win32'. It then prompts for help, copyright, credits, or license information. The user enters '>>>' three times, and the Python interpreter calculates the product of the two large numbers: '66666666666666666666*77777777777777777777' resulting in '5185185185185185179481481481481481482'.

图 3-1 程序运行结果

北京理工大学汇编语言实验报告

[illegible]

图 3-2 程序运行结果

[illegible]

图 3-3 程序运行结果

图 3-4 程序运行结果

4.1 实验目的

4.2 界面设计与界面编程代码展示:

界面设计的设计图如下所示:



图 4-1 浮点运算器界面设计

其中创建界面编程的代码如下所示：处理界面中窗口信息函数为_WinMain() 函数：

```
1.  local structWndClass: WNDCLASSEX
2.  local structMsg: MSG
3.  invoke GetModuleHandle, NULL ;获得相应程序的句柄
4.  mov H_app,eax ;存放应用程序的句柄
5.  invoke RtlZeroMemory, addr structWndClass, sizeof structWndClass
6.  invoke LoadCursor, 0, IDC_ARROW
7.  mov structWndClass.hCursor, eax
8.  push H_app
9.  pop structWndClass.hInstance
10.
11. mov structWndClass.cbSize, sizeof WNDCLASSEX
12. mov structWndClass.style, CS_BYTEALIGNWINDOW or CS_BYTEALIGNWINDOW
13. mov structWndClass.lpfnWndProc, offset _WinMain ;最关键的，设置窗口信息处理函数
14. mov structWndClass.cbClsExtra, 0
15. mov structWndClass.cbWndExtra,DLGWINDOWEXTRA
16. mov structWndClass.hbrBackground, COLOR_WINDOW + 1
17. mov structWndClass.lpszMenuName,NULL
18. mov structWndClass.lpszClassName, offset winClassName
19. invoke RegisterClassEx, addr structWndClass
```

```
20. invoke CreateWindowEx, WS_EX_CLIENTEDGE, offset winClassName,
    offset winCaptionName, WS_OVERLAPPEDWINDOW, 200, 250, 250, 350
    , NULL, NULL, H_app, NULL
21. mov H_win, eax ;存放窗口程序的句柄
22. invoke ShowWindow, H_win, SW_SHOWNORMAL
23. invoke UpdateWindow, H_win
```

在处理窗口获得信息的函数WinMain中，使用CreateWindowEx()函数添加了一些按钮和文本显示框，CreateWindowEx()函数的参数含义依次为：指定扩展窗口的样式，指定类名，窗口名，指定窗口样式，位置x，位置y，窗口宽度，窗口长度，父句柄，菜单句柄，将与窗口关联的模块的实例句柄和传递给窗口指针指向WM_CREATE消息的IParam参数。

创建的按钮和文本框位于.if eax == WM_CREATE消息下：

```
1. invoke CreateWindowEx, NULL, offset static, offset tips, WS_CH
    ILD or WS_VISIBLE, 5, 10, 300, 20, hWnd, 100, H_app, NULL
2.
3. invoke CreateWindowEx, WS_EX_RIGHT, offset static, offset Ou
    tput, WS_CHILD or WS_VISIBLE or WS_BORDER OR WS_SIZEBOX, 30, 50
    , 145, 50, hWnd, 90, H_app, NULL ; txt
4.
5. invoke CreateWindowEx, NULL, OFFSET button, OFFSET num0, WS_
    CHILD OR WS_VISIBLE, 30, 230, 55, 25, hWnd, 300, H_app, NULL ;
    0
6.
7. invoke CreateWindowEx, NULL, OFFSET button, OFFSET point, WS
    _CHILD OR WS_VISIBLE, 90, 230, 25, 25, hWnd, 12, H_app, NULL ;
    .
8. invoke CreateWindowEx, NULL, OFFSET button, OFFSET addop, WS
    _CHILD OR WS_VISIBLE, 120, 230, 25, 25, hWnd, 13, H_app, NULL
    ; +
9. invoke CreateWindowEx, NULL, OFFSET button, OFFSET equop, WS
    _CHILD OR WS_VISIBLE, 150, 230, 25, 25, hWnd, 17, H_app, NULL
    ; =
10.
11. invoke CreateWindowEx, NULL, OFFSET button, OFFSET num1, WS_
    CHILD OR WS_VISIBLE, 30, 200, 25, 25, hWnd, 301, H_app, NULL;1
12. invoke CreateWindowEx, NULL, OFFSET button, OFFSET num2, WS_
    CHILD OR WS_VISIBLE, 60, 200, 25, 25, hWnd, 302, H_app, NULL;2
13. invoke CreateWindowEx, NULL, OFFSET button, OFFSET num3, WS_
    CHILD OR WS_VISIBLE, 90, 200, 25, 25, hWnd, 303, H_app, NULL;3
```



```

14.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET subop, WS_
    _CHILD OR WS_VISIBLE, 120, 200, 25, 25, hWnd, 14, H_app, NULL
    ; -
15.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET sinop, WS_
    _CHILD OR WS_VISIBLE, 150, 200, 25, 25, hWnd, 18, H_app, NULL
    ; sin
16.
17.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET num4, WS_
    _CHILD OR WS_VISIBLE, 30, 170, 25, 25, hWnd, 304, H_app, NULL;4
18.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET num5, WS_
    _CHILD OR WS_VISIBLE, 60, 170, 25, 25, hWnd, 305, H_app, NULL;5
19.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET num6, WS_
    _CHILD OR WS_VISIBLE, 90, 170, 25, 25, hWnd, 306, H_app, NULL;6
20.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET mulop, WS_
    _CHILD OR WS_VISIBLE, 120, 170, 25, 25, hWnd, 15, H_app, NULL
    ; *
21.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET cosop, WS_
    _CHILD OR WS_VISIBLE, 150, 170, 25, 25, hWnd, 19, H_app, NULL
    ; cos
22.
23.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET num7, WS_
    _CHILD OR WS_VISIBLE, 30, 140, 25, 25, hWnd, 307, H_app, NULL;7
24.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET num8, WS_
    _CHILD OR WS_VISIBLE, 60, 140, 25, 25, hWnd, 308, H_app, NULL;8
25.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET num9, WS_
    _CHILD OR WS_VISIBLE, 90, 140, 25, 25, hWnd, 309, H_app, NULL;9
26.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET divop, WS_
    _CHILD OR WS_VISIBLE, 120, 140, 25, 25, hWnd, 16, H_app, NULL
    ; /
27.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET tanop, WS_
    _CHILD OR WS_VISIBLE, 150, 140, 25, 25, hWnd, 20, H_app, NULL
    ; tan
28.
29.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET back, WS_
    _CHILD OR WS_VISIBLE, 30, 110, 85, 25, hWnd, 21, H_app, NULL ;
    保留按钮，为了美观，之后优化可以为其添加同能
30.  invoke CreateWindowEx, NULL, OFFSET button, OFFSET ceop, WS_
    _CHILD OR WS_VISIBLE, 120, 110,

```

其中，可以看到，界面设计的顶部框（“计算器：…”）和显示数字结果的计算器显示框，是static类型（静态类型，不能改变）。其余的均是按钮类型，按下该按钮，窗口处理函数中eax将会收到WM_COMMAND信号，之后可以根据wParam的值对按下

的按钮进行不同的处理。

在《实验内容3：文本文件内容对比》也同样使用到了这些函数，对此不再进行赘述，使用函数以及对应的描述大抵相同。

4.3 程序整体思路：

计算器的整体实验主要是通过汇编语言浮点运算的函数来进行的。使用浮点运算函数：fadd, fsub, fmul, fdiv, fsin, fcos, ftanp来进行运算。

在整个计算器过程中，需要处理以下几项问题：

1. 如何对程序的按钮输入进行显示和提取：

完成该功能，通过按下按钮，在Output字符串进行赋值，使用SetDlgItemText函数使其在文本框中进行显示。在需要将其转换为数字时，使用sscanf函数，将Output字符串作为浮点数进行输入到变量中。

2. 如何完成对数和运算符的记录：

这里采用运算数栈和运算符栈对数和运算符进行记录。笔者的本意是打算完成将输入的一串算式转换成逆波兰式进行计算，但是之后笔者发现在汇编语言中进行优先级的区分过于麻烦，因此笔者未完成这项工作。但是运算数栈和运算符栈仍然在程序中起到了作用，可以完成程序的连续运算功能，即不需要每次按下等号“=”键才能完成运算，例如输入“3+3+3+3=”，程序可以计算出12的结果。但是注意，由于笔者没有对运算符的优先级进行区分，因此当输入“3+3*4-5”时，计算器会输出19的运算结果（即按照输入运算符的先后顺序计算）。

3. 如何进行运算：

在按下等号键或sin键cos键tan键时，分别使用其对应的浮点运算函数进行运算。将操作数栈的栈顶压入浮点栈中，然后进行根据浮点栈的内容使用fadd, fsub, fmul, fdiv, fsin, fcos, ftanp进行运算，运算结果输出到output字符串中，最后使用SetDlgItemText函数使其在文本框中进行显示。

4.4 程序中各个功能模块的介绍：

数字按钮的信息处理：

当按下数字按钮时，将数字组合放入字符串Output中（其中使用strlen函数判断Output串的长度，返回到寄存器eax中），之后将Output串通过SetDlgItemText()函

数显示在计算器的显示框中。

```
1. .elseif (eax <= ID_NUM9) && (eax >= ID_NUM0)
2.     sub eax, ID_NUM0
3.     add eax, 30h
4.     mov bl, al
5.     invoke strlen, addr Output
6.     .if (byte ptr[Output+eax-1] < '0') || (byte ptr[Output+eax-1] > '9')
7.     .if byte ptr[Output+eax-1] != '.'
8.         xor eax, eax
9.     .endif
10. .endif
11.
12.     mov byte ptr[Output+eax], bl
13.     mov byte ptr[Output+1+eax], 0
14.     invoke SetDlgItemText, hWnd, 90, offset Output
```

当按下小数点按钮时的信息处理：

处理同按下数字按钮的操作，只不过更简单一些，在这里需要注意的是。按下小数点时需要先判断字符串的最后一位是否是小数点，如果是小数点，那么应该让其按下的效果无效。

```
1. .elseif eax==12
2.     invoke strlen, addr Output
3.     .if (byte ptr[Output+eax-1] != '.')
4.         mov byte ptr[Output+eax], '.'
5.         mov byte ptr[Output+1+eax], 0
6.     .endif
7.     invoke SetDlgItemText, hWnd, 90, offset Output
```

+*/符号按钮的信息处理：

按下这些运算符时，首先通过sscanf函数，将Output中的字符串作为浮点数据进行输入，输入到一个存储浮点类型的变量中，将其入栈。之后在根据运算符的不同，将运算符的类型记录在运算符栈中。

这里需要阐述，为什么要采用运算符栈和运算数栈两个栈，而不直接进行存放计算。采用运算符栈和运算数栈的原因在于为了实现连续加减乘除的操作，比如，按下键：“3+3+4+2”，之后再按下等号键，运算数栈和运算符栈会将他们存储下来，在按下等号=键的时候，按照其中的栈的存放顺序将他们依次进行计算。这样，可以保证在按下+*/时也对其进行计算。

```
1. .elseif (eax>=13) && (eax<=16) ;+*/
```

```

2.      push eax
3.      mov ebx,[ntop]
4.      invoke sscanf,addr Output,addr formatStr,addr nstack[ebx*8]
5.      inc [ntop]                                ;运算符之前的操作数
        入栈
6.      .if [optop]>0
7.      fld qword ptr[nstack]
8.      .if byte ptr[opstack]=='+'
9.      fadd qword ptr[nstack+8]
10.     .elseif byte ptr[opstack]=='-'
11.     fsub qword ptr[nstack+8]
12.     .elseif byte ptr[opstack]=='*'
13.     fmul qword ptr[nstack+8]
14.     .else
15.     fdiv qword ptr[nstack+8]
16.     .endif
17.     fstp qword ptr[nstack]
18.     dec [ntop]
19.     dec [optop]
20.     .endif
21.     pop eax
22.     .if eax==13
23.     mov byte ptr[Output], '+'
24.     .elseif eax==14
25.     mov byte ptr[Output], '-'
26.     .elseif eax==15
27.     mov byte ptr[Output], '*'
28.     .else
29.     mov byte ptr[Output], '/'
30.     .endif
31.
32.     mov byte ptr[Output+1],0
33.     mov al,byte ptr [Output]
34.     mov byte ptr [opstack],al
35.     inc [optop]
36.     invoke SetDlgItemText, hWnd, 90, offset Output

```

sin cos tan按钮的信息处理:

这些按钮的处理比较简单，同样通过sscanf函数将文本Output中的字符作为数据输入到存储数据的变量中，之后将其入浮点数栈。这时，不需要再记录操作的类型，直接判断是sin cos 还是 tan，调用fsin fcos函数即可。在调用ftanp函数时，遇

到了一些问题导致结果永远为1, 分析了很久也没有处理好, 所以在处理tan运算时, 我选择使用其sin值除以其cos值。

```
1. .elseif (eax>=18) && (eax<=20) ;sin cos tan
2.     push eax
3.     mov esi,[ntop]
4.     invoke sscanf,addr Output,addr formatStr,addr nstack[esi*8]
5.     fld qword ptr nstack[esi*8]
6.     pop eax
7.     .if eax==18
8.         fsin
9.     .elseif eax==19
10.        fcos
11.    .elseif eax==20
12.        fcos
13.        fstp nstack[esi*8+8]
14.        fld nstack[esi*8]
15.        fsin
16.        fdiv nstack[esi*8+8]
17.    .endif
18.    fstp qword ptr nstack[esi*8]
19.    invoke sprintf,addr Output,addr formatStr2,nstack[esi*8]
20.    invoke SetDlgItemText, hWnd, 90, offset Output
```

当按下‘=’时的信息处理:

按下等于号后, 再次将文本中输入的浮点数进行读取, 读取后根据符号存储类型的不同判断是+*/的哪一类符号, 进行对应的浮点运算即可。

```
1. .elseif eax==17 ;=
2.     mov ebx,[ntop]
3.     invoke sscanf,addr Output,addr formatStr,addr nstack[ebx*8]
4.     inc dword ptr [ntop] ;=之前的
    操作数入栈
5.     .if dword ptr [optop]>0
6.         fld qword ptr[nstack]
7.         .if byte ptr[opstack]=='+'
8.             fadd qword ptr[nstack+8]
9.         .elseif byte ptr[opstack]=='-'
10.            fsub qword ptr[nstack+8]
11.        .elseif byte ptr[opstack]=='*'
12.            fmul qword ptr[nstack+8]
13.        .else
14.            fdiv qword ptr[nstack+8]
```

```
15.      .endif
16.      fstp qword ptr[nstack]
17.      mov dword ptr [ntop],0
18.      mov dword ptr [optop],0
19.      .endif
20.      invoke sprintf,addr Output,addr formatStr2,qword ptr [nstack]
21.      invoke SetDlgItemText, hWnd, 90, offset Output
```

4.5 程序运行结果

程序运行结果截图如下图所示：

使用加法计算 $3+4.1+5.2$ ：

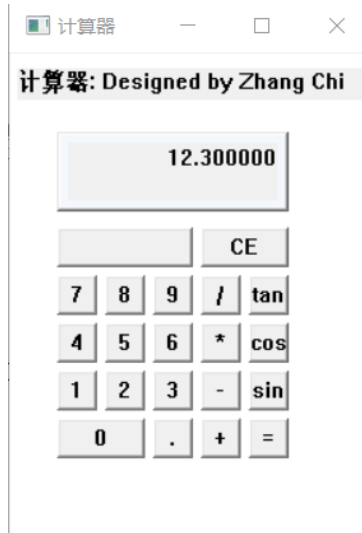


图 4-2 程序运行结果

使用减法计算 $12.5-5.6$ ：

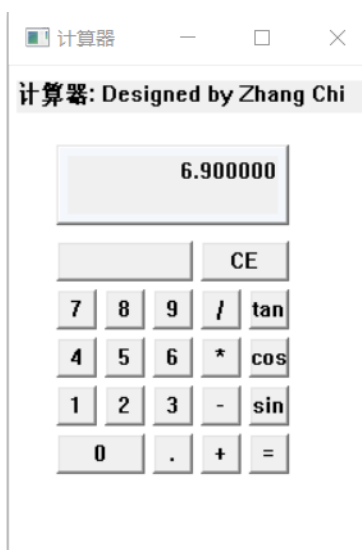


图 4-3 程序运行结果

使用乘法计算 $3.1 * 4.5$:



图 4-4 程序运行结果

使用除法计算 $9.3 / 3.1$ ，结果如下图所示：

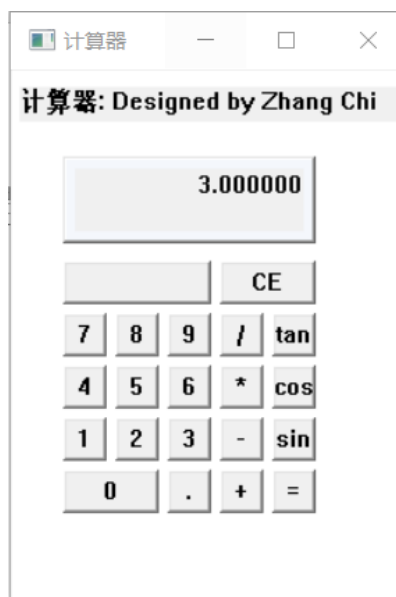


图 4-5 程序运行结果

使用sin操作计算 $\sin 3.14$ （弧度制），得出结果接近于0，正确



图 4-6 程序运行结果

使用cos操作计算cos3.14（弧度制），得出结果接近于0，正确



图 4-7 程序运行结果

使用tan操作计算tan(1.57)（弧度制），得出结果接近于正无穷，正确



图 4-8 程序运行结果

经验证，这些结果均是正确的结果。

第 5 章 实验内容 3：文本文件内容对比

5.1 实验目的

文本文件内容对比，要求以Windows界面风格实现两个文本文件内容的对比。若两文件内容一样，输出相应提示；若两文件不一样，输出对应的行号。

5.2 程序设计的基本流程以及界面设计

程序设计的流程如下所示：

该程序需要实现文本文件的逐行比较，使用CompareFile()函数。该文件生成的结果保存在differs中。在读取文本文件时，要同时逐行读取两个文件的内容，两个文本文件的行数同时增加。在逐行读取文本文件时，会遭遇以下三种情况：

(1) 两个文件的该行均读取到缓冲区中，均有数据，使用C库函数strcmp对两个文件的字符串进行对比，如果相同则继续，如果不相同则将两个文件的行数按照格式的要求保存在differs中，等待输出。

(2) 两个文件中，一个文件读取到该行，另外一个文件未读取到该行，此时直接将该行按照格式的要求保存在differs中，等待输出

(3) 两个文件均未读取到该行，说明两个文件读取完毕，结束对比

读取文本文件的一行数据，使用ReadLine()函数。该函数在读取文本文件的其中一行时，如果读取到换行字符‘\n’，那么说明该行读取完毕，如果读取到文件终止字符<EOF>，说明文件读取完毕。

界面设计如下图所示：

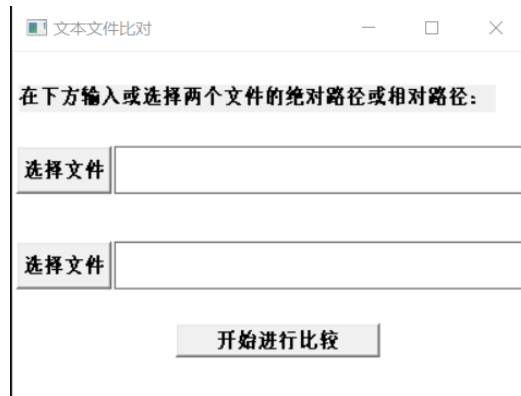


图 5-1 文本文件对比界面设计

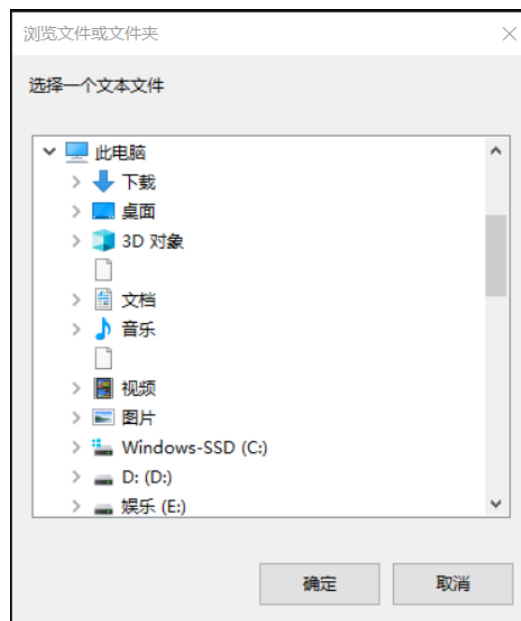


图 5-2 文本文件对比界面设计

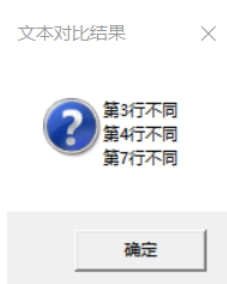


图 5-3 文本文件对比界面设计

其中包含有一个static静态控件类作为输入的提示, 包含有两个编辑框, 用于输入, 可以直接输入文件的绝对路径或相对路径。在编辑框左侧有两个“选择文件”按钮, 点击后可以浏览电脑中的文件, 选择确定后可以生成其绝对路径在右侧的编辑框中。最下方的“开始进行比较”按钮, 用于开始对两个文本文件进行比较, 点击确定后即可看到比较结果。比较结果的提示使用MessageBox来提示。

5.3 程序核心模块的介绍

5.3.1 文件对比模块 CompareFile:

```
1. CompareFile proc fpath1:ptr byte, fpath2:ptr byte
2.     local lp1 :dword
3.     local lp2 :dword
4.     local index_line :dword
5.     local buffer_differ[1024] :byte
6.     MOV errorflag, 1
7.     invoke CreateFile, fpath1, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN
        _EXISTING, FILE_ATTRIBUTE_NORMAL, NULL
8.     mov fp1, eax
9.     ;判断路径是否出错
10.    .if fp1 == 0
11.        invoke MessageBox, NULL, offset file1error, offset errortitle, MB_O
            K + MB_ICONQUESTION
12.        MOV errorflag, 0
13.        JMP ENDFUNC
14.    .endif
15.    invoke CreateFile, fpath2, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN
        _EXISTING, FILE_ATTRIBUTE_NORMAL, NULL
16.    mov fp2, eax
17.    .if fp2 == 0
18.        invoke MessageBox, NULL, offset file2error, offset errortitle, MB_O
            K + MB_ICONQUESTION
19.        MOV errorflag, 0
20.        JMP ENDFUNC
21.    .endif
22.    mov differ_num, 0
23.    mov index_line, 0
24.    mov esi, offset differs
25.    mov byte ptr[esi], 0
26.
27. CMP0:
```

```
28. inc index_line
29. invoke ReadLine, fp1, offset buffer1
30. mov lp1, eax
31. invoke ReadLine, fp2, offset buffer2
32. mov lp2, eax
33.
34. CMP1:
35. cmp lp1, 0
36. jne CMP3
37. cmp lp2, 0
38. jne CMP2
39. jmp ENDFUNC;都等于0 就 return
40.
41. CMP2:
42. invoke sprintf, addr buffer_differ, offset DiffContent, index_line
43. invoke strcat, offset differs, addr buffer_differ
44. inc differ_num
45. jmp CMP0
46.
47. CMP3:
48. cmp lp2,0
49. jne CMP4
50.
51. invoke sprintf, addr buffer_differ, offset DiffContent, index_line
52. invoke strcat, offset differs, addr buffer_differ
53. inc differ_num
54. jmp CMP0
55.
56. CMP4:
57. ;都不等于0 时, 使用 strcmp 函数对文件进行对比
58. invoke strcmp, offset buffer1, offset buffer2
59. cmp eax, 0
60. je CMP0
61. invoke sprintf, addr buffer_differ, offset DiffContent, index_line
62. invoke strcat, offset differs, addr buffer_differ
63. inc differ_num
64. jmp CMP0
65.
66. ENDFUNC:
67. ret
68. CompareFile endp
```

使用CreateFile函数根据文件路径打开文件, 然后判断文件句柄是否存在, 如果

不存在则报错。打开文件句柄后，使用ReadLine函数读取第index_line行。之后判断读取的文件行的长度，根据如上的程序流程，判断该行是否相同。

5.3.2 读取一行模块 ReadLine:

```
1. ReadLine proc stdcall uses ebx, fileHand: HANDLE, lpLineBuf: ptr  
   r byte  
2.     ; 读完文件 eax 返回值为 0  
3.     LOCAL br:DWORD  
4.     LOCAL char:BYTE  
5.     mov ebx, lpLineBuf  
6.  
7. L2:  
8.     invoke ReadFile, fileHand, addr char, 1, ADDR br, NULL  
9.  
10.    .if br == 0  
11.    jmp L1  
12.    .endif  
13.    mov al, char  
14.    mov [ebx], al  
15.    inc ebx  
16.  
17.    .if char == 0AH  
18.    jmp L1 ;判断是否为换行\n 为换行就 return  
19.    .elseif  
20.    jmp L2  
21.    .endif  
22. L1:  
23.    xor al, al  
24.    mov [ebx], al  
25.    invoke strlen, lpLineBuf  
26.    ret  
27. ReadLine endp
```

ReadLine函数中使用的ReadFile函数，将文件的一行读取到缓冲区中。之后判断字符，如果遇到文件终止符，则返回。如果判断到为换行字符，同样说明读取完毕，返回。返回时将读取的长度保存在寄存器EAX中。

5.4 程序运行结果

程序运行结果如下所示。

下图为直接使用输入相对路径的方式比较两文件是否存在不同。

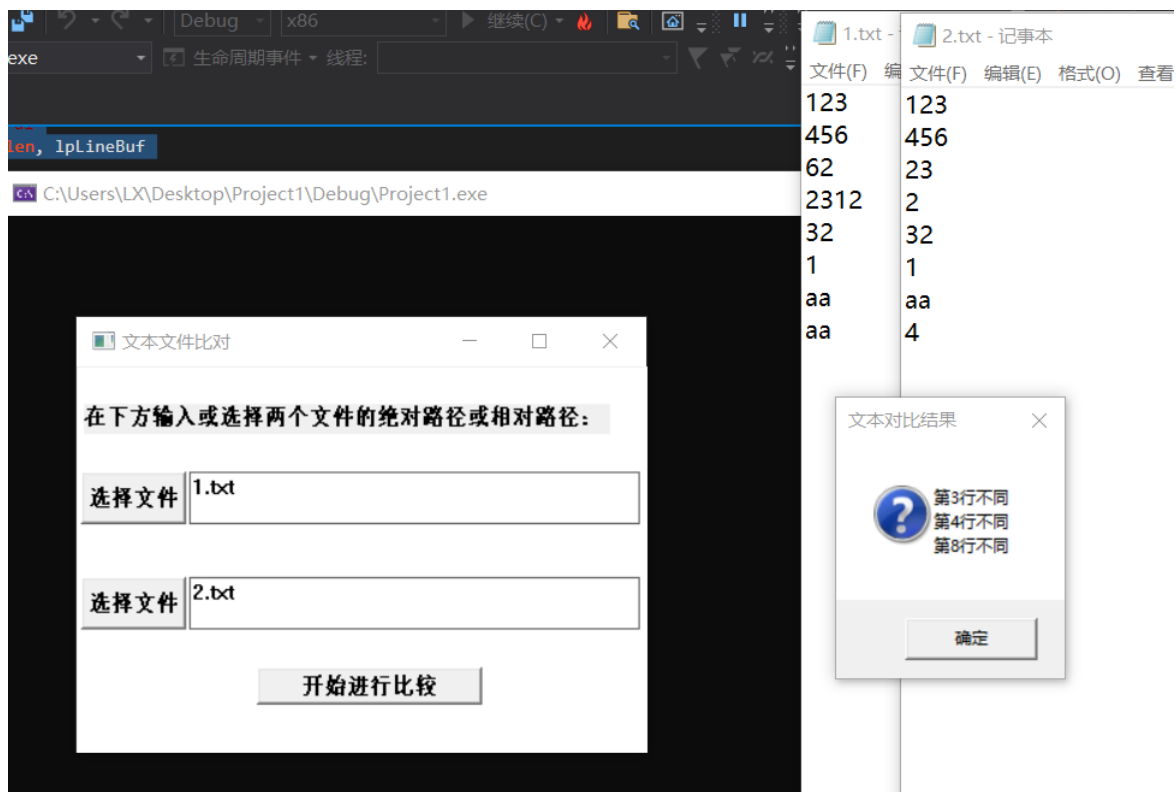


图 5-4 程序运行结果

下图为使用选择文件按钮，选中文件并获取文件的绝对路径来比较文件。

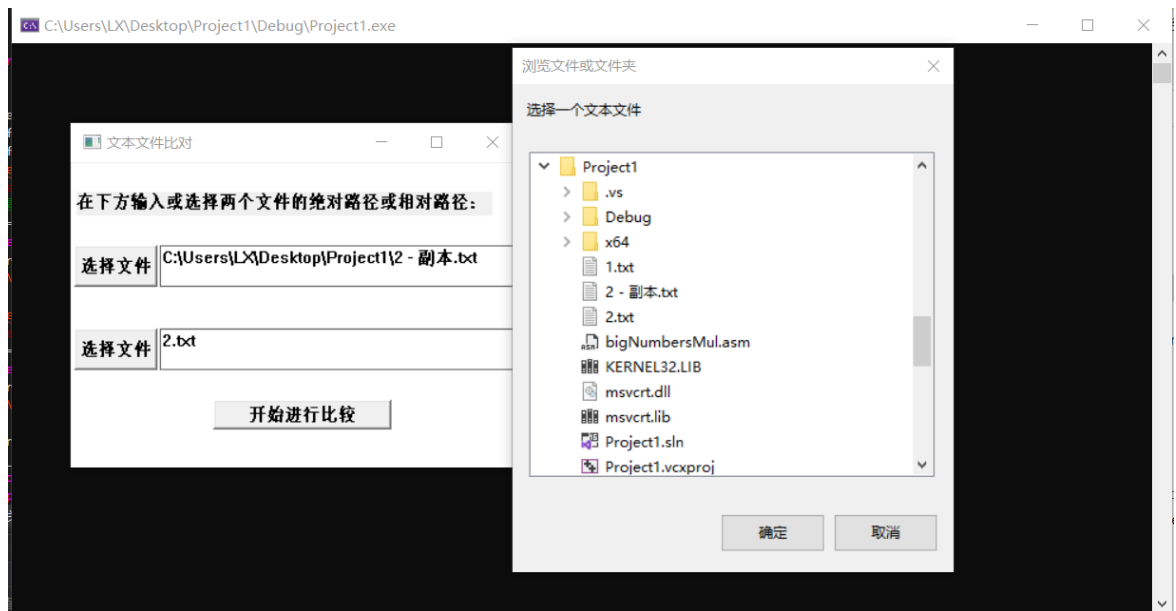


图 5-5 程序运行结果

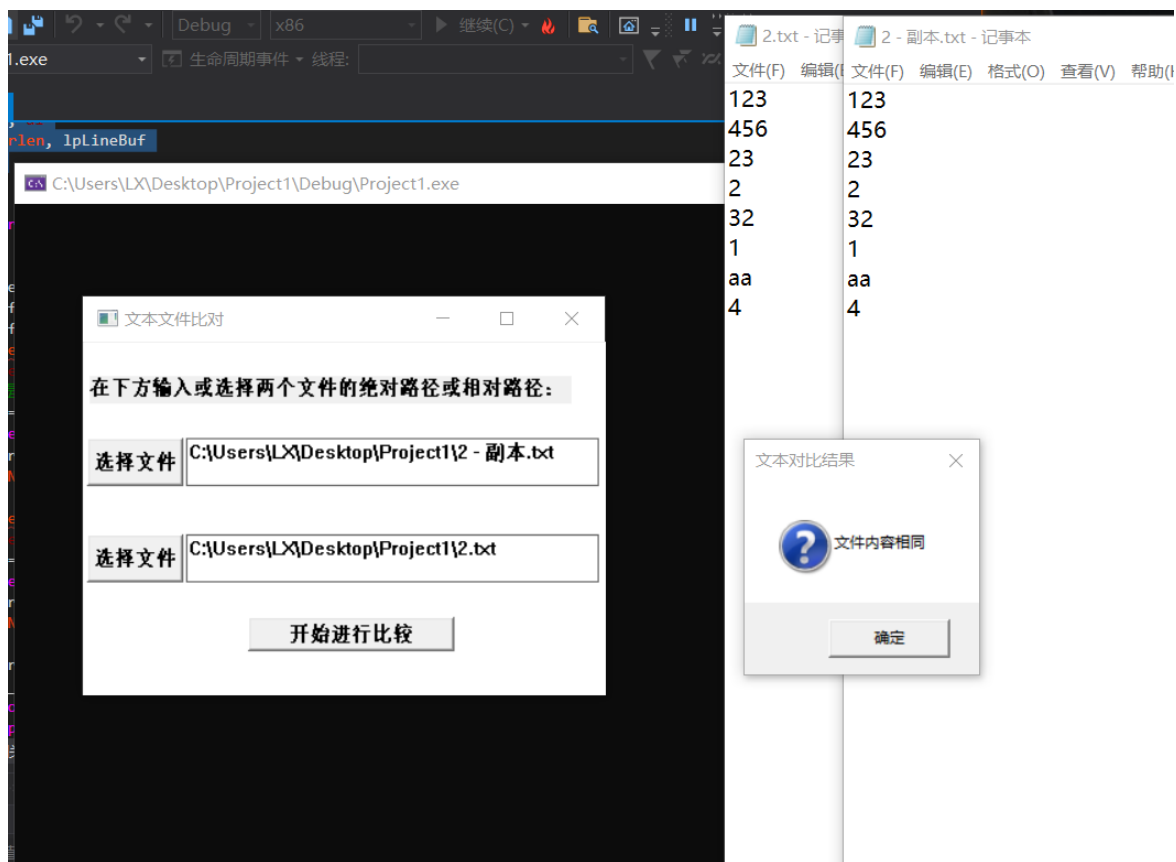


图 5-6 程序运行结果

两次比较的返回结果均正确且符合要求。

第 6 章 实验心得体会

在本次实验中，本人完成了大数相乘，浮点计算器以及文本文件比对三个方面的汇编语言实验。在本次汇编实验中，本人了解了各类基础汇编指令的使用，参考了网络上的大量文档，理解了windows界面编程。

在难度方面，本人首先完成了大数相乘，其中不包含Windows界面编程的相关指令。之后，本人按照实验顺序完成了浮点计算器的汇编编程，使用了浮点运算指令并对Windows界面编程进行了熟悉，参考了网络上的文档，学会了Windows界面编程的相关函数和操作。最后，本人实现了文本文件对比，对Windows界面编程和汇编语言的相关指令有了进一步的熟悉。

致 谢

值此实验完成，我要感谢老师对关于汇编语言知识的悉心讲解。同时，我要感谢网络上的相关资源发布者，让我进一步理解了汇编语言界面编程的相关操作。