Experiment Report 3

1. Experimental requirements and objective

a) Be able to code, assemble, and execute a program with Visual C++ and MASM.
b) Know how to link your programs to an external code library.
c) Know how to create conditional and looping structures using assembly language.

2. Experimental environment

a) Hardware environment

The microcomputer CPU more than Pentium, more than 120GB capacity hard drive, more than 1GB of memory.

b) Software environment

Visual Studio 2008 and above versions of applications.

3. Experimental contents

1) Prime number program

a) Write a procedure named IsPrime that sets the Zero flag if the 32-bit integer passed in the EAX register is prime.
b) Write a test program that prompts the user for an integer, calls IsPrime, and displays a message indicating whether or not the value is prime. Continue prompting the user for integers and calling IsPrime until the user enters -1.

2) Str_remove Procedure

a) Write a procedure named **str_remove1** that removes *n* characters from a string. Pass a pointer to the position in the string where the characters are to be removed. Pass an integer specifying the number of characters to remove. The following code, for example, shows how to remove "xxxx" from **target**:

```
.data
    target BYTE "abcxxxxdefghijklmop",0
.code
    INVOKE str_remove1, ADDR [target+3], 4
```

b) Write a procedure named **str_remove2** that removes *n* characters from a string. Pass an integer specifying the position in the string from where the characters are to be removed. Pass an integer specifying the number of characters to remove. The following code, for example, shows how to remove "xxxx" from **target**:

```
.data
    target BYTE "abcxxxxdefghijklmop",0
```

```
.code
    INVOKE str_remove2, ADDR target, 4, 4
```

c) Write a procedure named **main** that displays the string before and after removing characters.

3) Bubble Sort

a) Write a procedure named **BubbleSort** to perform a bubble sort on a 32-bit signed integer array whose offset address and count of elements are passed by parameters.

b) Write a procedure named **main** that creates an array of randomly ordered 32-bit integers, and display the ordered integers after calling the **BubbleSort** procedure.

4. Experiment Result

(1) Prime number:

```
Please input a number:1
It's not a Prime number
Please input a number:2
It's a Prime number
Please input a number:3
It's a Prime number
Please input a number:4
It's not a Prime number
Please input a number:5
It's a Prime number
Please input a number:6
It's not a Prime number
Please input a number:7
It's a Prime number
Please input a number:8
It's not a Prime number
Please input a number:9
It's not a Prime number
Please input a number:10
It's not a Prime number
Please input a number:-1
Press any key to continue...
```

```
Please input a number:29
It's a Prime number
Please input a number:148
It's not a Prime number
Please input a number:332
It's not a Prime number
Please input a number:350
It's not a Prime number
Please input a number:21
It's not a Prime number
Please input a number:158
It's not a Prime number
Please input a number:251
It's a Prime number
Please input a number:291
It's not a Prime number
Please input a number:306
It's not a Prime number
Please input a number:25
It's not a Prime number
Please input a number:-1
Press any key to continue...
```

```
Please input a number:324
It's not a Prime number
Please input a number:301
It's not a Prime number
Please input a number:32
It's not a Prime number
Please input a number:151
It's a Prime number
Please input a number:146
It's not a Prime number
Please input a number:19
It's a Prime number
Please input a number:261
It's not a Prime number
Please input a number:446
It's not a Prime number
Please input a number:415
It's not a Prime number
Please input a number:75
It's not a Prime number
Please input a number:-1
Press any key to continue...
```

(2) Str_remove procedure

(expected)

Str_remove1:

```
abcdefghijklmop
```

Str_remove2:

```
abcdefghijklmop
```

(3) Bubble sort

```
Input 10 numbers: 36
0
24
28
4
8
16
12
20
32
Before sorting: +0 +4 +8 +12 +16 +20 +24 +28 +32 +36
After sorting: +0 +4 +8 +12 +16 +20 +24 +28 +32 +36
Press any key to continue...
```

5. Source Code of Programs

(1) Prime Procedure

```
title Prime
INCLUDE Irvine32.inc

.data
    flag DWORD        0
    pW       BYTE "It's a Prime number",0
    npW     BYTE "It's not a Prime number",0
    hint    BYTE "Please input a number:",0
.code
Prime PROC,
    val:DWORD
    mov      flag,    0
    mov      ecx, 1
    cmp      val, -1
    jz       L1
L0:
    .while ecx <= val
        mov edx, 0
        mov  eax,    val
        div  ecx
        .if edx == 0
            add flag,    1
        .endif
        inc ecx
    .endw
L1:
    ret
Prime ENDP

main proc
Lp1:
    mov      edx, offset hint
    call     WriteString
    call     ReadInt
    push     eax
    call     Prime
    mov      ebx, flag
    .if      flag == 0
    ret
```

```
    .endif
    cmp     ebx, 2
    jz      L2
    jnz     L3

L2:
    mov     EDX, offset pW
    call WriteString
    jmp     L4
L3:
    mov     EDX, offset npW
    call WriteString
    jmp     L4

L4:
    mov     flag,   0
    call    Crlf
    call Lp1

L5:
    call WaitMsg
    ret
main endp
end main
```

(2) Str_remove

```
title Str_Remove
INCLUDE Irvine32.inc

.data
    target   BYTE "abcxxxxdefghijklmop",0
    Size BYTE lengthof target
    len     BYTE ?
.code
Remove1 PROC,
    pos:PTR BYTE,num1:WORD

    mov  bl,      byte ptr [pos]
    sub  bl,      byte ptr [target]
    mov  si, 0
    .while si < bl
        mov     al, target[si]
        call WriteChar
```

```
        inc     si
    .endw
    add si,  num1
    .while   si < Size
        mov     al, target[si]
        call WriteChar
        inc     si
    .endw
    ret

Remove1 endp

Remove2 PROC,
    pos:PTR BYTE, num1:WORD,num2:WORD

    mov  si,  0
    .while si < num1
        mov     al,  target[si]
        call WriteChar
        inc     si
    .endw
    add  si,  num2
    .while si < num2
        mov     al,target[si]
        call WriteChar
        inc     si
    .endw
    ret

Remove2 endp
main PROC
    INVOKE   Remove1, ADDR[target+3], 4
    call Crlf
    INVOKE   Remove2, ADDR target, 4, 4
    call Crlf
main endp
end main
```

(3) Bubble

```
title Bubble
INCLUDE Irvine32.inc

.data
```

```
    Data1    DWORD    10    Dup(?)
    temp1    DWORD    0
    temp2    DWORD    0
    hint BYTE "Input 10 numbers: ",0
    input    BYTE "Before sorting: ",0
    output   BYTE "After sorting: ",0
.code

Bubble   PROC,
    Data: PTR DWORD
    mov      edx, offset input
    call WriteString
    mov      ebx,0
    mov      ecx,0
    .while   ecx < 10
        mov      eax,Data
        add      eax,ebx
        call WriteInt
        mov      al,' '
        call WriteChar
        inc      ecx
        add      ebx,4h
    .endw
    mov  ecx,0
    mov  esi,0
    mov  eax,0
    mov  edx,0
    .while ecx < 10
        .while   esi <    ecx
            mov  ebx, Data
            add  ebx, edx
            mov  edi, Data
            add edi, eax
            .if  ebx <    edi
            mov edx,edi
            mov  eax,ebx
            .endif
            inc  esi
            add eax,4h
        .endw
        mov  eax,0
        mov  esi,0
        inc ecx
        add edx,4h
```

```
    .endw
    mov  ecx,0
    mov  ebx,0
    mov eax,0
    call Crlf
    mov      edx, offset output
    call WriteString
    .while ecx < 10
        mov      eax, Data
        add      eax, ebx
        call WriteInt
        mov      al,' '
        call WriteChar
        inc      ecx
        add      ebx,4h
    .endw
    call Crlf
    ret
Bubble endp

main PROC
    mov      edx,offset hint
    call WriteString
    mov      ecx, 0
    mov      ebx, 0
    .while   ecx < 10
        call ReadInt
        mov      ebx, Data1
        add      ebx, eax
        inc      ecx
        add      ebx,4h
    .endw
    push Data1
    call Bubble
    call WaitMsg
main endp
end  main
```

6. Summary

- How to define a procedure and then correctly call it.

Long before, we have already learnt that the format of a procedure without passing parameters is that "(procedure name) PROC … (procedure name) endp". This time, the passing parameters are needed. And the format is as follows:

(procedure name) PROC, ;This comma is important
(parameter name) : (parameter type), …, (parameter name) : (parameter type)

Also, when calling a procedure, notice that whether passing parameters are matched with the define type. Otherwise, the whole program cannot pass the compilation.

- Two types of calling a procedure

The first type is pushing the parameters into the stack, then use the "call" keyword. And the second type is, use the "invoke" keyword with the parameters behind.

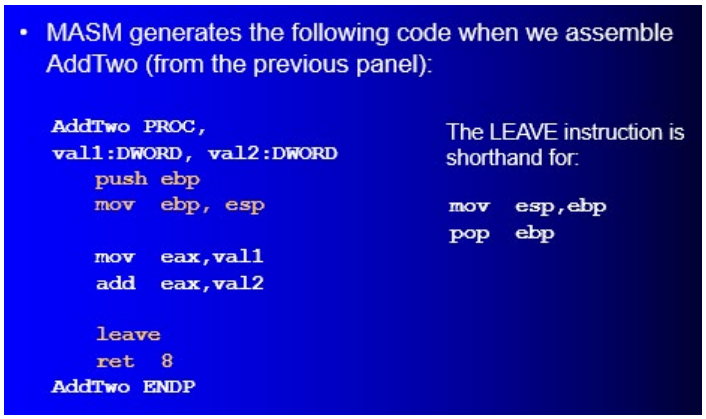
After this experiment, the most impressive idea is, if the number of parameters is not too much, use the "push" and "call", or else, choose to use the "invoke" keyword.

- Classification of address and content

Notice that the common register can store both the address and content, it's quite easy to mixture them. Thus, the strategy is store the address in common register, for example, store in the "eax" accumulator, then [eax] is the value stored in the address and can be operated.

- Module

Actually, we code with the pseudocode, and it helps us done lots of things like implicitly push and so on. The most classical example is as follows:



- At the end

Sincerely thanks for teaching, maybe it's a bit hard for us to absorb all the content of the course. However, teacher always use the examples to help us to get a better understanding after learning the theory with great patience. Once again, thanks for the teaching.