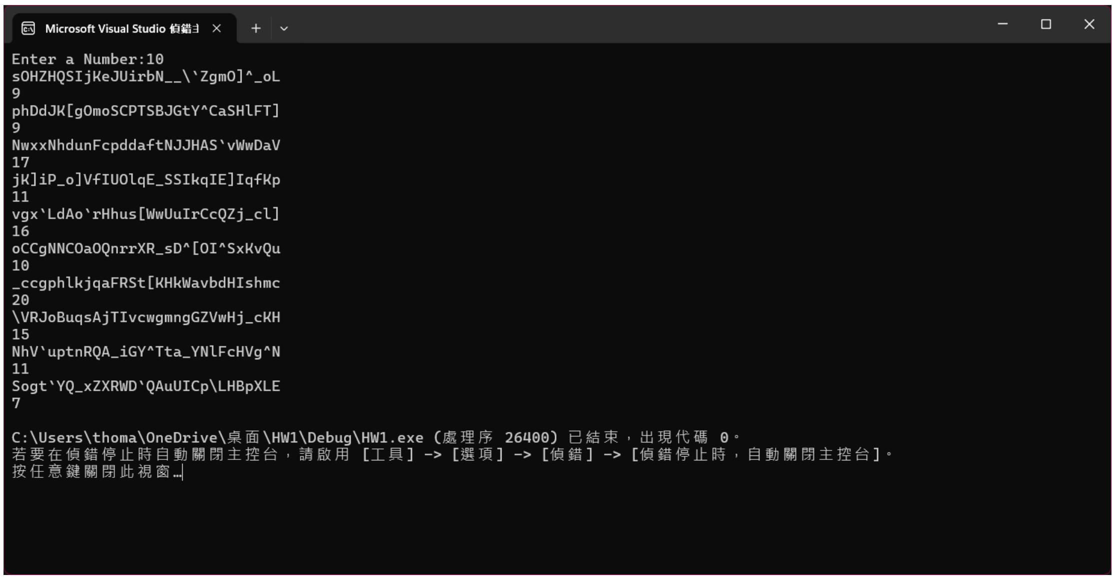
1.

(a)screenshot



```
Enter a Number:10
sOHZHQSIjKeJUirbN__\`ZgmO]^_oL
9
phDdJK[gOmoSCPTSBJGtY^CaSHlFT]
9
NwxxNhdunFcpddaftNJJHAS`vWwDaV
17
jK]iP_o]VfIUOlqE_SSIkqIE]IqfKp
11
vgx`LdAo`rHhus[WwUuIrCcQZj_cl]
16
oCCgNNCOaOQnrrXR_sD^[OI^SxKvQu
10
_ccgphlkjqaFRSt[KHkWavbdHIshmc
20
\VRJoBuqsAjTIvcwgmngGZVwHj_cKH
15
NhV`uptnRQA_iGY^Tta_YNlFcHVg^N
11
Sogt`YQ_xZXRWD`QAuUICp\LHBpXLE
7

C:\Users\thoma\OneDrive\桌面\HW1\Debug\HW1.exe (處理序 26400) 已結束，出現代碼 0。
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。
按任意鍵關閉此視窗…
```

(b)code

```
INCLUDE Irvine32.inc

.data
    arrayStr    BYTE 30 DUP (?)                     ; define an array to store random array string
    countArray EQU 30                               ; count random array string element
    outputStr  BYTE "Enter a Number:", 0            ; output string

.code
main PROC
    mov EDX, OFFSET outputStr
    call WriteString                                ; print output string
    call ReadInt                                    ; read the input number
    mov ECX, EAX                                    ; move the input number to ECX for outer loop counter

L1: call Rand1                                      ; generate random array string
    call CheckSmallLetter                           ; check the small letter in the random array string
    loop L1
    exit
main ENDP
```

```
Rand1 PROC
   push ECX                              ; store outer loop counter to stack
   mov ECX, countArray                          ; load inner loop counter to ECX
   mov ESI, OFFSET arrayStr                       ; load arrayStr index to esi

L2:
   mov EAX,56
   call RandomRange                           ; generate random int
   add EAX, 65                            ; start from 41h equals to 65d
   mov [ESI], AL                           ; move the generate random int into arrayStr
   inc ESI                           ; increase esi
   call WriteChar                         ; print the character
   loop L2

   call Crlf                         ; change line after print all the character
   pop ECX                              ; restore outer loop counter to ECX
   ret
Rand1 ENDP

CheckSmallLetter PROC
   push ECX                              ; store outer loop counter to stack
   mov ECX, countArray                          ; load inner loop counter to ECX
   mov ESI, OFFSET arrayStr                       ; load arrayStr index to esi
   xor EAX, EAX                            ; initialize the lowercase count to 0

L3:
   movzx EBX, byte ptr [ESI]                   ; load the character into ebx
   inc ESI                           ; increase esi
   cmp BL, 61h                          ; compare with 'a'
   jb L3_end                           ; if less than 'a', skip
   cmp BL, 7Ah                          ; compare with 'z'
   ja L3_end                           ; if greater than 'z', skip
   inc EAX                             ; increment the lowercase count
L3_end:
   loop L3

   call WriteDec                          ; print the result of small letter
   call Crlf                         ; change line after print all the character
```

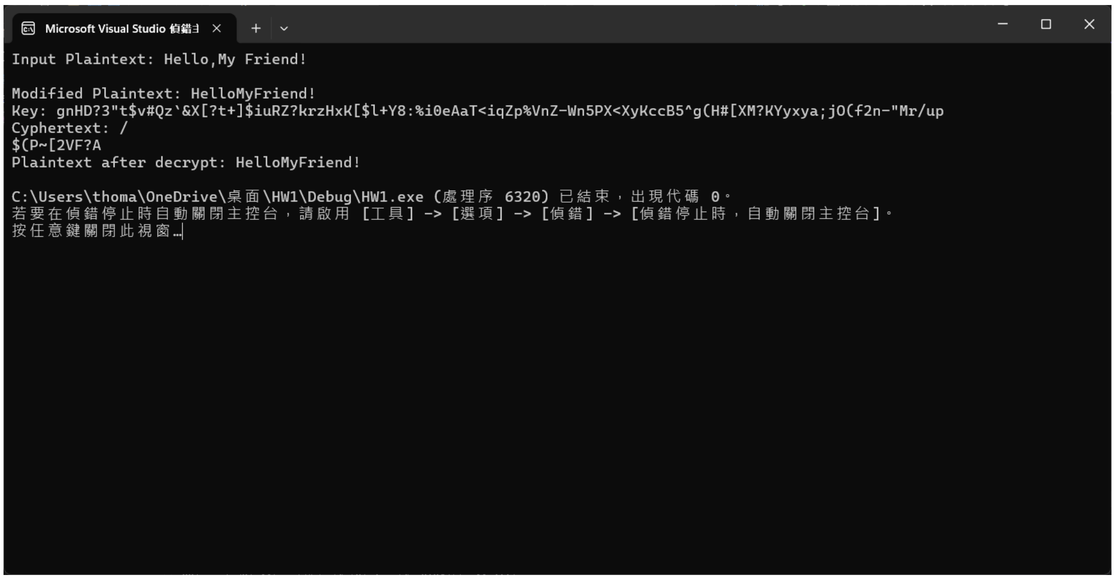
```
    pop ECX                              ; restore outer loop counter to ECX
    ret
CheckSmallLetter ENDP

END main
```

(c)explain code detailly

首先，讓使用者輸入需要產生 隨機字串的次數，再利用輸入值執行迴圈，迴圈內含有兩個 Procedure，分別為 Rand1，生成隨機字串用，以及 CheckSmallLetter，用於檢查隨機字串中所含的小寫字母數量，在 Rand1 中，我依照產生字串的長度來設定迴圈的次數，並且在迴圈中，我設定了 RandomRange 的範圍，並且添加適當的偏移量使得生成的數字皆在題目所要求的範圍之內，之後便利用指標的方式將生成出來的數字存入字串中，最後待迴圈結束後，利用 WriteChar 印出字串中的 ASCII code，結束這個 procedure call; 另一個 procedure 為 CheckSmallLetter，在這個 procedure 中，迴圈設定一樣為字串的長度，在迴圈中，我利用 jump above 和 jump below 來確定是否是小寫以外的範圍，如果是，則不計數，如果不是，則計數，最後再將結果利用 WriteDec 印出，並結束這個 procedure，除此之外，所有的 procedure 都會將原在迴圈外的 ECX 之數值存入 stack 中，避免 procedure 和 main 之間相互影響。

2.

(a)screenshot



```
Input Plaintext: Hello,My Friend!

Modified Plaintext: HelloMyFriend!
Key: gnHD?3"t$v#Qz`&X[?t+]$iuRZ?krzHxK[$l+Y8:%i0eAaT<iqZp%VnZ-Wn5PX<XyKccB5^g(H#[XM?KYyxya;jO(f2n-"Mr/up
Cyphertext: /
$(P~[2VF?A
Plaintext after decrypt: HelloMyFriend!

C:\Users\thoma\OneDrive\桌面\HW1\Debug\HW1.exe (處理序 6320) 已結束，出現代碼 0。
若要在偵錯停止時自動關閉主控台，請啟用 [工具] -> [選項] -> [偵錯] -> [偵錯停止時，自動關閉主控台]。
按任意鍵關閉此視窗…
```

(b)code

```
INCLUDE Irvine32.inc
BUFMAX = 128                              ;maxium buffer size

.data
    inputPlaintext BYTE "Input Plaintext: ", 0
    modifiedPlaintext BYTE "Modified Plaintext: ", 0
    key BYTE "Key: ", 0
    cyphertext BYTE "Cyphertext: ", 0
    plaintextAfterDecrypt BYTE "Plaintext after decrypt: ", 0
    buffer BYTE BUFMAX+1 DUP(0)                    ; the reason that setting the buffer
size to BUFMAX+1 is because null terminator occupy one bytes
    modifiedBuffer BYTE BUFMAX+1 DUP(0)
    keySeed BYTE BUFMAX+1 DUP(0)
    bufSize DWORD ?
    keySize DWORD ?

.code
main PROC
    call    InputTheString
    call    ModifyTheString
    call    GenerateKey
    call    EncryptTheString
    call    DecryptTheString
    exit
main ENDP

InputTheString PROC
    pushad                                  ; store all registers to stack before starting
    mov EDX,OFFSET inputPlaintext
    call    WriteString                     ; print "Input Plaintext: "
    mov ECX,BUFMAX                          ; maximum character count
    mov     EDX,OFFSET buffer               ; point to the buffer
    call    ReadString                      ; input the buffer
    mov     bufSize,EAX                     ; save the buffer length to bufSize
    call Crlf                               ; next line
    popad                                   ; restore all registers to stack after using
    ret
InputTheString ENDP

ModifyTheString PROC
    pushad                                  ; store all registers to stack before starting
    mov ECX,bufSize                         ; loop bufSize
    mov ESI,OFFSET buffer                   ; load buffer starting address to ESI
```

```
   mov EDI,OFFSET modifiedBuffer                ; load modifiedBuffer starting
address to EDI
   xor EBX,EBX                               ; clear EBX to zero and using it to count the
modifiedBuffer size
L1: mov AL,[ESI]                             ; load buffer character into AL
   cmp AL,20h                               ; compare with 'Space'
   je L1_end                               ; jump taken if equals to 'Space'
   cmp AL,2Ch                               ; compare with ','
   je L1_end                               ; jump taken if equals to ','
   cmp AL,2Eh                               ; compare with '.'
   je L1_end                               ; jump taken if equals to '.'
   mov [EDI],AL                             ; if not taken, put the character to new string
   inc EDI                                ; increase EDI
   inc EBX                                 ; increase EBX
L1_end:
   inc ESI                                ; increase ESI
   loop L1

   mov BYTE PTR [EDI],0                          ; add null terminator at the end of
keySeed
   inc EBX
   mov bufSize,EBX                              ; store EBX to bufSize
   mov EDX,OFFSET modifiedPlaintext
   call    WriteString                        ; print "Modified Plaintext: "
   mov EDX,OFFSET modifiedBuffer
   call    WriteString                        ; print modifiedBuffer
   call    Crlf                              ; next line
   popad                                 ; restore all registers to stack after using
   ret
ModifyTheString ENDP

GenerateKey PROC
   pushad                                ; store all registers to stack before starting
   mov ESI,OFFSET keySeed                        ; load keySeed index to ESI
   mov EAX,BUFMAX                              ; setting key range
   call    RandomRange
   inc EAX
   mov keySize,EAX                             ; store key size
```

```
    mov ECX,EAX                          ; loop keySize
    xor EAX,EAX                          ; clear EAX to zero
L2: mov AL,5Bh                           ; the range is start from 21h to 7Dh
    call    RandomRange                  ; generate random int
    add AL,21h                           ; start from 21h
    mov [ESI],AL                         ; move character to ESI
    inc ESI                              ; increase ESI
    inc EBX                              ; increase EBX
    loop L2

    mov BYTE PTR [ESI], 0                ; add null terminator at the end of
keySeed
    mov EDX, OFFSET key
    call WriteString                     ; print key
    mov EDX, OFFSET keySeed
    call WriteString                     ; print keySeed
    call Crlf
    popad                                ; restore all registers to stack after using
    ret
GenerateKey ENDP

EncryptTheString PROC
    pushad                               ; store all registers to stack before starting
    mov ESI,OFFSET modifiedBuffer        ; load modifiedBuffer address to ESI
    mov EDI,OFFSET keySeed               ; load keySeed starting address to ESI
    mov EAX,EDI                          ; store keySeed starting address to EAX
    mov ECX,bufSize                      ; loop bufsize
L3: mov BL,[ESI]                         ; load modifiedBuffer character into AL
    cmp BL,0
    je L3_end
    mov BH,[EDI]                         ; load keySeed character into AH
    xor BL,BH                            ; encrypt modifiedBuffer
    mov [ESI],BL
    inc ESI
    inc EDI
    mov EAX, EDI                         ; store the current EDI value in EAX
    sub EAX, OFFSET keySeed              ; calculate the offset from the start of
keySeed
```

```
    cmp EAX, keySize                        ; compare with the keySize
    jb L3
    mov EDI, OFFSET keySeed
    jmp L3
L3_end:
    mov EDX, OFFSET cyphertext
    call WriteString                        ; print "Cyphertext: "
    mov EDX, OFFSET modifiedBuffer
    call WriteString                        ; print modifiedBuffer
    call Crlf                               ; next line
    popad                                   ; restore all register to stack after using
    ret
EncryptTheString ENDP

DecryptTheString PROC
    pushad                                  ; store all registers to stack before starting
    mov ESI, OFFSET modifiedBuffer                 ; load modifiedBuffer address to ESI
    mov EDI, OFFSET keySeed                        ; load keySeed starting address to EDI
    mov EAX, EDI                            ; store keySeed starting address to EAX
    mov ECX, bufSize                        ; loop bufSize
L4: mov BL, [ESI]                           ; load modifiedBuffer character into BL
    cmp BL, 0
    je L4_end
    mov BH, [EDI]                           ; load keySeed character into BH
    xor BL, BH                              ; decrypt modifiedBuffer
    mov [ESI], BL                           ; store the decrypted byte back
    inc ESI
    inc EDI
    mov EAX, EDI                            ; store the current EDI value in EAX
    sub EAX, OFFSET keySeed                        ; calculate the offset from the start of
keySeed
    cmp EAX, keySize                        ; compare with the keySize
    jb L4                                   ; if less than keySize, continue the loop
    mov EDI, OFFSET keySeed                        ; reset EDI to the start of keySeed
    jmp L4
L4_end:
    mov EDX, OFFSET plaintextAfterDecrypt
    call WriteString                        ; print "Plaintext after decrypt: "
    mov EDX, OFFSET modifiedBuffer
    call WriteString                ; print decrypted modifiedBuffer
    call Crlf                       ; next line
    popad                           ; restore all registers to stack after using
    ret
DecryptTheString ENDP

END main
```

(c)explain code detailly

第二個問題相較於第一個問題複雜一些，我分為五個 Procedure 來執行此程式，分別如下：一、InputTheString，用於輸入字串，方法參考於輔助教學平台範例程式碼；二、ModifyTheString，用於從輸入字串中去除題目所要求之特殊字元；三、GenerateKey，用於生成加密金鑰的種子碼；四、EncryptTheString，利用重複加密種子碼與去除特殊字元後的字串進行 XOR 運算；五、DecryptTheString，基本上就是 EncryptTheString 的逆操作，全部的 Procedure 皆使用 pushad 和 popad 來防止 procedure 之間的 register 互相影響。其中先來說明 ModifyTheString，其運行方式和第一題的 CountSmallLetter 有些像，都是數字的比較，不同的地方在於我們這邊額外宣告了一個空間，來存放不是特殊字元的字元，依此建立修改後的字串，並利用 EBX 來計算修改後的字串長度；在 GenerateKey 中，我們首先用 RandomRange 來生成金鑰種子碼的長度並記錄，接下來利用生成出來的字串長度來跑迴圈，這邊的作法很像第一題中生成隨機字串的部分，不再次贅述；EncryptTheString 中，我們利用修改後的字串以及金鑰種子碼來生成 CypherText，並依照修改後的字串長度來做為迴圈的次數，迴圈引入了兩個檢查機制，一 、若修改後的字元為 0，則代表已經到了 null terminator，即跳出迴圈，如果金鑰種子碼到盡頭，則回到起始位置，所以這也是程式碼中記錄金鑰種子碼起始位置的原因，帶迴圈結束，CypherText 也就此完成；最後，DecryptTheString 就只是 EncryptTheString 的逆向操作。