Lab 5: Input/Output using a Library of Procedures

Contents

5.1 Using an External Library of Procedures for Input and Output

The *Irvine32* library, authored by Kip Irvine, provides a number of useful procedures that can be used for basic input and output. First, we need to define few terms:

- **Console:** This is the Command Prompt window running in color text mode under MS-Windows. The console can execute 32-bit protected-mode programs as well as 16-bit real-address mode programs. The console window is divided into rows and columns. There are 25 rows and 80 columns by default, but it can be resized.
- **Standard Output:** by default, the console defines standard output as the screen display device, although standard output can be also redirected to write to a file.
- **Standard Input:** By default, the console defines standard input as the keyboard device, although standard input can be also redirected to read from a file.

Here is the list of output procedures for writing characters, strings, and integers. You can find more details in the textbook and class notes. There is also an online help for the Irvine library that comes with this lab.

Procedure	Description
Clrscr	Clears the console and locates the cursor at the upper left corner.
Crlf	Writes end-of-line control chars (CR=13, LF=10) to standard output.
WriteChar	Writes a single character in register AL to standard output.
WriteString	Writes a null-terminated string, with address in EDX, to standard output.
WriteHex	Writes register EAX in hexadecimal format to standard output.
WriteBin	Writes register EAX in binary format to standard output.
WriteDec	Writes register EAX in unsigned decimal format to standard output.
WriteInt	Writes register EAX in signed decimal format to standard output.

5.2 Writing Characters, Strings, and Integers to Standard Output

```
TITLE Writing characters, strings, and integers  (output.asm)

; Testing Following Procedures in the assembly32.lib Library:

; Clrscr:       Clears the console screen
; Crlf:         Write CR and LF characters (end-of-line)
; WriteChar:    Write a single character to standard output
; WriteString:  Write a null-terminated string to standard output
; WriteHex:     Write 32-bit integer in eax in hexadecimal format
; WriteBin:     Write 32-bit integer in eax in binary format
; WriteDec:     Write 32-bit integer in eax in unsigned decimal format
; WriteInt:     Write 32-bit integer in eax in signed decimal format

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data

CR  EQU     0Dh         ; carriage return
LF  EQU     0Ah         ; line feed

string  BYTE "Hello World",CR,LF,0

.code
main PROC
; Clear the screen
    call Clrscr         ; Call procedure Clrscr

; Write a character to standard output
    mov  al, 'A'        ; al = 'A' (or 41h)
    call WriteChar      ; Write character in register al
    call Crlf           ; Write CR and LF chars (end-of-line)

; Write a null-terminated string to standard output
    lea  edx, string    ; load effective address of string into edx
    call WriteString    ; write string whose address is in edx

; Write an integer to standard output
    mov  eax,0F1A37CBFh ; eax = 0F1A37CBFh
    call WriteHex       ; Write eax in hexadecimal format
    call Crlf           ; Write CR and LF chars (end-of-line)

    call WriteBin       ; Write eax in binary format
    call Crlf           ; Write CR and LF chars (end-of-line)

    call WriteDec       ; Write eax in unsigned decimal format
    call Crlf           ; Write CR and LF chars (end-of-line)

    call WriteInt       ; Write eax in signed decimal format
    call Crlf           ; Write CR and LF chars (end-of-line)

    exit
main ENDP
END main
```

5.2.1 Lab Work: Assemble and Link Output.asm

5.2.2 Lab Work: Trace the Execution of Output.exe

Guess the Console Output of the above program and write it in the specified box.

Console for 'output.exe'

Run the 32-bit Windows Debugger. Open the source file *output.asm* from the **File** menu if it is not already opened. Watch the registers by selecting **Registers** in the **View** menu or by pressing **Alt+4**. Customize the registers to have registers **al**, **eax**, and **edx** on top of the list.

Place the cursor at the beginning of *main* procedure and press **F7** to start debugging it. Press **F10** to step through the execution of the program. Watch the changes in the output window. Make the necessary corrections to the Console Output that you have guessed.

5.3 Displaying Memory/Registers and Setting Text Color

The following output procedures are used to dump a block of memory, to dump registers, to set the foreground and background colors of text, to display a wait message, and to locate the cursor at a specific row and column. For more details, refer to your textbook, class notes, and online help for the Irvine library.

Procedure	Description
DumpMem	Writes a block of memory to standard output in hexadecimal format. The start address of the memory block should be in the ESI register The number of elements to be displayed should be in the ECX register The type of each element should be in the EBX register
DumpRegs	Writes the EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EFLAGS, and EIP registers in hexadecimal, as well as the Carry, Sign, Zero, and Overflow flags.
SetTextColor	Sets the foreground and background colors of all subsequent text output. The desired color should be specified in the EAX register before calling this procedure. The least significant 4 bits of EAX specify the text color and the next 4 bits specify the background color.
WaitMsg	Writes "Press [Enter] to continue…" and waits for Enter to be pressed.
Gotoxy	Locate the cursor at row specified by DH and column specified by DL.

The following program demonstrates the use of the above procedures:

```
TITLE Setting Text Color, Dumping Memory and Registers  (Output2.asm)

; Testing following Output Procedures in the assembly32.lib Library:

; Clrscr:       Clears the console screen
; SetTextColor: Set the foreground and background colors of text
; DumpMem:      Write a block of memory in hexadecimal
; DumpRegs:     Display basic registers and flags in hexadecimal
; WaitMsg:      Display a message and wait for Enter key to be pressed
; Gotoxy:       Put the cursor at a specific row/column on the console

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
CR  EQU     0Dh                     ; carriage return
LF  EQU     0Ah                     ; line feed

string   BYTE    "This is a string", CR, LF, 0

.code
main PROC
; Clear the screen after setting text color
    mov  eax, yellow+(blue*16)  ; yellow = 14 and blue = 1
    call SetTextColor           ; set yellow text on blue background
    call Clrscr                 ; Call procedure Clrscr

; Call DumpMem that display a block of memory to standard output
    mov  esi, OFFSET string     ; esi = address of memory block
    mov  ecx, LENGTHOF string   ; ecx = number of elements to display
    mov  ebx, TYPE BYTE         ; ebx = type of each element
    call DumpMem                ; write 19 bytes of string

; Call WaitMsg that displays "Press [Enter] to continue ..."
    call WaitMsg                ; wait for [Enter] key to be pressed

; Call DumpRegs that display the basic registers and flags in hex
    call DumpRegs               ; write basic registers

; Call WaitMsg after locating the cursor on the console
    mov  dh, 10                 ; row 10
    mov  dl, 20                 ; column 20
    call Gotoxy                 ; locate cursor
    call WaitMsg                ; wait for [Enter] key to be pressed

    exit
main ENDP
END main
```

5.3.1 Lab Work: Assemble and Link Output2.asm

5.3.2 Lab Work: Trace the Execution of Output2.exe

Run the 32-bit Windows Debugger. Open the source file *output2.asm* from the **File** menu if it is not already opened. Watch the registers by selecting **Registers** in the **View** menu or by pressing **Alt+4**. Place the cursor at the beginning of *main* procedure and press **F7** to start debugging it. Press **F10** to step through the execution of the program. Watch the changes in

the registers as well as the output window. When the program displays the Wait message "**`Press [Enter] to continue...`**" it is waiting for your input. Click inside the Console output and press Enter. The program will advance to the next instruction. Go back to the program and click **F10** to continue tracing it. Write below the final Console output:

Console for 'output2.exe'

You can also run the program *Output2.exe* from a command prompt to check its output.

5.4 Reading Characters, Strings, and Integers from Standard Input

The following input procedures are used to read characters, integers, and strings:

Procedure	Description
ReadChar	Reads a character from standard input and returns it in AL register. The character is NOT echoed on the screen.
ReadString	Reads a string from standard input. Reading stops when the user presses the [Enter] key. Before calling *ReadString*, EDX should contain the address of the array of bytes where the input characters should be stored, and ECX should contain the maximum number of bytes to be read plus one extra byte for the null character to terminate the string. This procedure returns a count of the actual number of bytes read in the EAX register.
ReadHex	Reads a hexadecimal string from standard input, converts it to a number, and returns it in the EAX register. Reading stops when the user presses the [Enter] key. A maximum of eight hex digit chars should be entered. The hex digits: **0-9**, **a-f**, and **A-F** may be used only. Leading spaces are not allowed. No error checking is performed and no error messages are displayed. If an invalid hex string is entered then it will not be converted properly.
ReadInt	Reads a signed integer string from standard input, converts it to a number, and returns it in the EAX register. Reading stops when the user presses the [Enter] key. An optional leading + or – character can be entered by the user. Decimal digits **0-9** can be entered only after the optional sign. Error checking is performed by this procedure. The overflow flag is set and an error message is displayed if the entered number is invalid or out-of-range and cannot be represented as a signed 32-bit integer.

The following program demonstrates the use of the above procedures:

```
TITLE Reading characters, strings, and integers  (input.asm)
; Testing Following Procedures in the assembly32.lib Library:
; ReadChar:     Read a single character from standard input
; ReadString:   Read a null-terminated string from standard input
; ReadHex:      Read hexadecimal integer from standard input
; ReadInt:      Read signed decimal integer from standard input

.686
.MODEL flat, stdcall
.STACK
INCLUDE Irvine32.inc

.data
charvar     BYTE    0           ; Character variable
string      BYTE    21 DUP(0)   ; Extra byte for null char
bytecount   DWORD   0           ; Count of bytes read in string
hexvar      DWORD   0           ; Unsigned integer variable
intvar      SDWORD  0           ; Signed integer variable

prompt1     BYTE    "Enter a character (char will not appear): ",0
prompt2     BYTE    "Enter a string (max 20 chars): ",0
prompt3     BYTE    "Enter a hexadecimal number (max 8 digits): ",0
prompt4     BYTE    "Enter a decimal number with optional sign: ",0

.code
main PROC
    call Clrscr

; Display prompt1
    lea  edx, prompt1
    call WriteString
; Read a character (without echo) from standard input
    call ReadChar           ; character is returned in AL
    mov  charvar, al        ; save character in charvar
    call Crlf               ; Write end-of-line after reading character

; Display prompt2
    lea  edx, prompt2
    call WriteString
; Read a null-terminated string from standard input
    lea  edx, string        ; edx = address of storage area for string
    mov  ecx, SIZEOF string ; ecx = max characters to be stored
    call ReadString         ; read string from standard input
    mov  bytecount, eax     ; eax = actual number of chars read

; Display prompt3
    lea  edx, prompt3
    call WriteString
; Read a hexadecimal string and convert it to a number
    call ReadHex            ; number is returned in EAX register
    mov  hexvar, eax        ; save number in hexvar

; Display prompt4
    lea  edx, prompt4
    call WriteString
; Read a signed decimal string and convert it to a number
    call ReadInt            ; number is returned in EAX register
    mov  intvar, eax        ; save number in intvar
    exit
main ENDP
END main
```

5.4.1 Lab Work: Assemble and Link Input.asm

5.4.2 Lab Work: Trace the Execution of Input.exe

Run the 32-bit Windows Debugger. Open the source file *input.asm* from the **File** menu if it is not already opened. Watch the registers by selecting **Registers** in the **View** menu or by pressing **Alt+4**. Customize the registers to have registers **al**, **eax**, **ecx**, **edx** on top of the list. Select **Memory** from the **View** menu, or press **Alt+5** to view memory. Type *string* in the virtual address box to view the *string* variable in memory. Select **Watch** from the **View** menu, or press **Alt+2** to open the Watch window. Watch the values of the variables *charvar*, *bytecount*, *hexvar*, and *intvar* by inserting their names.

Place the cursor at the beginning of *main* procedure and press **F7** to start debugging it. Press **F10** to step through the execution of the program.

When the program asks to **Enter a character (char will not appear):** then enter character **A** in the console window then press **F10** to step through **call ReadChar** in the program. When the program asks to **Enter a string (max 20 chars):** then enter **My String** and press enter. When the programs asks to **Enter a hexadecimal number (max 8 digits):** then enter **ab09F** and press enter. When the program asks to **Enter a decimal number with optional sign:** then enter **–12345678** and press enter.

Show the values of the following variables, just before exiting the program:

1) charvar (in hex and as a character) =												
2) 12 bytes of string (in hex)												
3) bytecount (in decimal) =												
4) hexvar (in hex) =												
5) intvar (in decimal) =												

5.4.3 Lab Work: Invalid Input

Repeat the program execution entering invalid input. For the string, enter a string longer than 20 characters. For the hexadecimal number, enter more than 8 digits and try also invalid characters. For the signed integer, try long integers that are outside the range –2,147,483,648 to +2,147,483,647. Also try invalid input and see what happens.

What happens when entering a string longer than 20 characters?

What happens when entering an invalid hexadecimal number?

What happens when entering an invalid decimal number?

5.5 Generating Random Numbers and Delaying Program Execution

The following miscellaneous procedures are defined in the *Irvine32.lib* library and can be of practical use to some programs:

Procedure	Description
Random32	Generates a 32-bit pseudorandom integer in the range 0 to FFFFFFFFh, and returns it in the EAX register. When called repeatedly, Random32 generates a pseudorandom sequence of integers.
RandomRange	Generates a pseudorandom integer from 0 to *n* – 1. Before calling *RandomRange*, *n* should be passed in the EAX register. RandomRange returns its result in the EAX register.
Randomize	Seeds the random number generators *Random32* and *RandomRange* with a unique value. The seed is initialized from the time of the day obtained from the system. This procedure virtually ensures that each time you run a program, the sequence of random integers will be different. You need to call *Randomize* once at the beginning of your program.
Delay	Delay program execution for specified *n* milliseconds. The register EAX should be set to the desired value of *n* before calling the *Delay* procedure.
GetMseconds	Return the number of milliseconds that have elapsed since midnight. The return value is in the EAX register. This procedure is useful when you want to measure the time between two events such as the beginning and end of program execution.

The following program illustrates the use of the above procedures:

```
TITLE Generating random numbers and delaying execution (rand.asm)

; Testing Following Procedures in the assembly32.lib Library:
; Randomize:   Seeds the random number generator with a unique value
; Random32:    Generates 32-bit random integer between 0 and FFFFFFFFh
; RandomRange: Generates a random integer between 0 and n-1
; Delay:       Delay program execution for specified n milliseconds
; GetMseconds: Number of milliseconds that have elapsed since midnight

.686
.MODEL flat, stdcall
.STACK

INCLUDE Irvine32.inc

.data
CR  EQU     0Dh         ; carriage return
LF  EQU     0Ah         ; line feed

rand1   BYTE    "Generating 5 pseudo-random integers ",
                "between 0 and FFFFFFFFh", CR, LF, 0
rand2   BYTE    "Generating 5 pseudo-random integers ",
                "between 0 and 999", CR, LF, 0
time    BYTE    "Execution time in milliseconds: ",0
start   DWORD   ?   ; start execution time
```

```
.code
main PROC
; Get starting execution time
    call GetMseconds        ; EAX = number of msecs since midnight
    mov  start, eax         ; save starting execution time

; Seeds the random number generator from the time of the day
    call Randomize          ; different seed for each run

; Display message rand1
    mov  edx, OFFSET rand1
    call WriteString

; Generate 5 random integers between 0 and FFFFFFFFh.
; Put a delay between each.
    mov  ecx,5              ; loop counter
L1: mov  eax, 1000          ; 1000 milliseconds
    call Delay              ; pause for 1 second
    call Random32           ; EAX = random integer
    call WriteHex           ; display ax hexadecimal integer
    call Crlf               ; advance cursor to next line
    Loop L1

; Display message rand2
    mov  edx, OFFSET rand2
    call WriteString

; Generate 5 random integers between 0 and 999.
; Put a delay between each.
    mov  ecx,5              ; loop counter
L2: mov  eax, 1000          ; 1000 milliseconds
    call Delay              ; pause for 1 second
    mov  eax,1000           ; indicate top of range + 1
    call RandomRange        ; EAX = random integer
    call WriteDec           ; display as unsigned decimal
    call Crlf               ; advance cursor to next line
    Loop L2

; Compute and display execution time
    mov  edx, OFFSET time
    call WriteString
    call GetMseconds        ; EAX = number of msecs since midnight
    sub  eax, start         ; difference since starting time
    call WriteDec

    exit
main ENDP
END main
```

5.5.1 Lab Work: Assemble and Link Rand.asm

5.5.2 Lab Work: Run Rand.asm

Run the *rand.exe* program from the Command Prompt twice and watch the console output. What changes need to be done to the *rand.asm* program to generate the **same random sequence** every time the program is executed?

Review Questions

1. Which procedure in the Irvine link library displays "Press [Enter] to continue …"?

2. Which procedure writes an integer in unsigned decimal format to standard output?

3. Which procedure generates a random integer within a selected range?

4. Which procedure places the cursor at a specific console window location?

5. What are the required input parameters for the *ReadString* Procedure?

6. Locate and examine the *Irvine.inc* file. What type of statements are inside this file?

Programming Exercises

1. Write a program that uses a loop to input ten signed 32-bit integers from the user, stores the integers in an array, and redisplays the integers.

2. Write a program that displays the string "Assembly Language is COOL" in four different colors. Each word should be displayed in a different color of your choice.

3. Write a program that clears the screen, places the cursor near the middle of the screen, prompts the user for two integers, adds the integers, and displays their sum.

4. Write a program that generates and displays 50 random integers between -20 and +20.

5. Write a program that generates and displays twenty random strings, each consisting of ten random capital letters {A .. Z}.

6. Write a program that displays a single character '*' at 100 random screen locations. Use a delay of 100 milliseconds before displaying the '*' at the next random screen location.

