**HW : Theme: Strings and Interrupts**

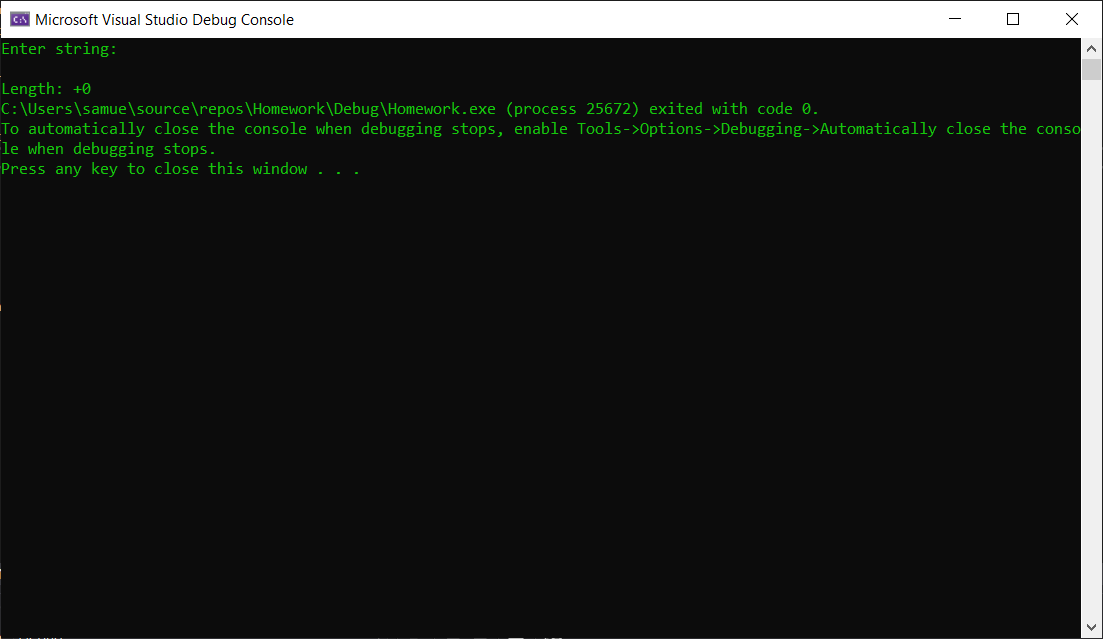
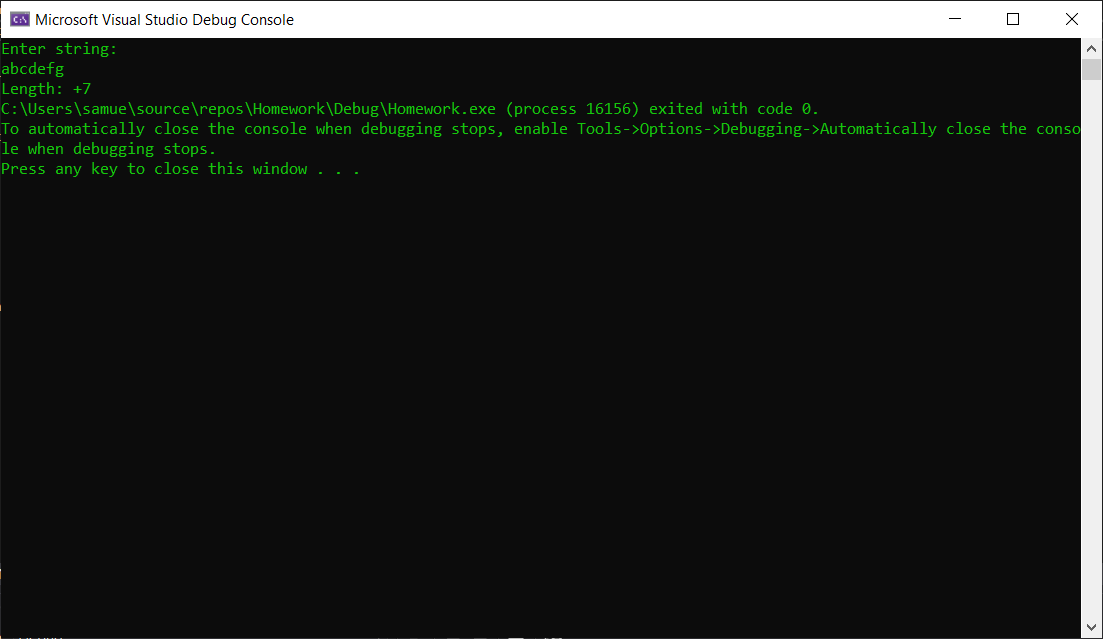
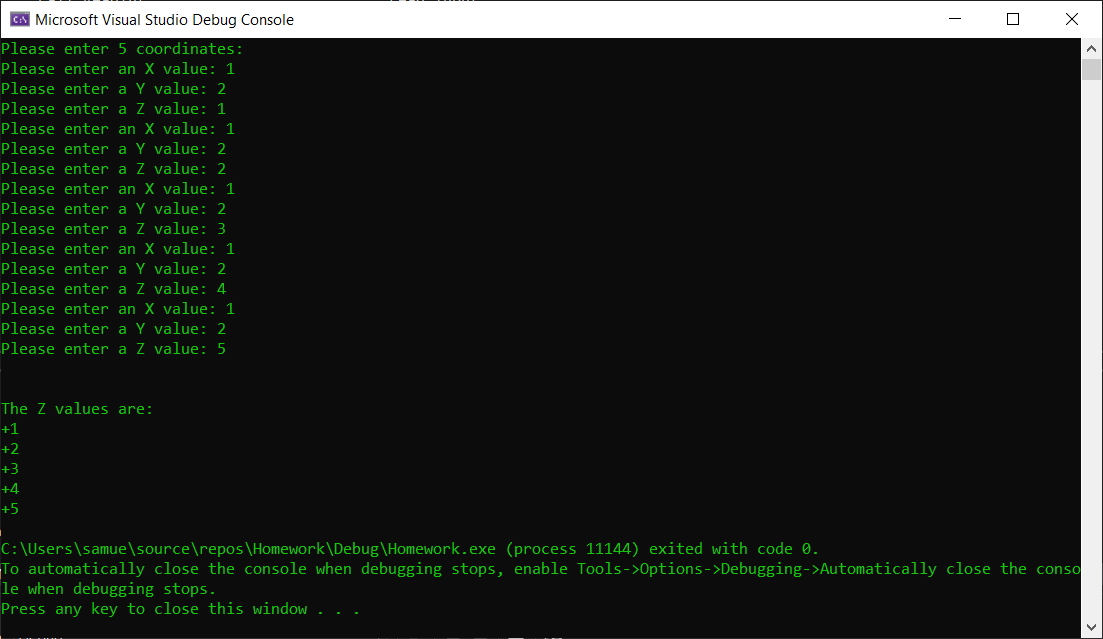
*(All main questions carry equal weight.  Credit awarded to only those answers for which work has been shown.)*

1. [Interrupts] What are hardware and software interrupts? Give examples of each.  What are maskable and non-maskable interrupts?  Provide examples of each.

Hardware and software interrupts interrupt the current process. When a interrupt is called, it executes an interrupt handler. RESET is a hardware interrupt from the on/off button, it clears register flags and directs boot routines to the operating system. INTR is from external devices, that is triggered by a handshake connection. INT is a software interrupt that invokes an interrupt handler and causes a specific event. TRAP is triggered by exceptions. Maskable interrupts are interrupts that can be disabled, such as INTR. A non-maskable interrupt is an interrupt that occurs within hardware and cannot be disabled, such as divide by zero errors, exceptions, power on/off button and NMI.

1. [Interrupts] Explain the process of interrupt vectoring.

Interrupt vectoring happens when an interrupt is invoked. The call address is transferred to a Interrupt Vector Table, which holds a 32 bit offset address, this address is located at \*4 the original call, and transfers to the interrupt handler. The interrupt handler processes the interrupt and then it is returned.

1. [Strings] Write a program that computes the number of characters in any string.  Test the robustness of your program using different strings including those of size 0.
2. .386
3. .model flat,stdcall
4. .stack 4096
5. ExitProcess proto, dwExitCode:dword
6. include Irvine32.inc
7. stringlength PROTO, pString: PTR BYTE
8. .data
10. stringQ byte "Enter string: ",0
11. lengthQ byte "Length: ",0
12. string BYTE 101 DUP(0)
13. .code
14. main proc
15. mov edx, OFFSET stringQ
16. call WriteString
17. call crlf
18. mov edx, OFFSET string
19. mov ecx, SIZEOF string
20. call ReadString
21. invoke stringlength, addr string
22. mov edx, OFFSET lengthQ
23. call WriteString
24. call WriteInt
25. main endp
26. stringlength proc uses edi,
27. pString: PTR BYTE
28. mov edi, pString
29. mov eax, 0
30. L1:
31. cmp byte ptr [edi], 0
32. je L2
33. inc edi
34. inc eax
35. jmp L1
36. L2: ret
37. stringlength endp
38. END main
39. [Structures] Using the structure example discussed in the book and slides, write a program that displays the *z*-coordinates of several points given as an array of coordinates in the data segment. Unlike the example in the book you should use 3-dimensional points. Test your program with various *(x, y, z)* Use base-indexed addressing to implement the program.  

.386

.model flat,stdcall

.stack 4096

ExitProcess proto, dwExitCode:dword

INCLUDE Irvine32.inc

points = 5

HYPERCUBE STRUCT

X SWORD ? ; 00

Y SWORD ? ; 02

Z SWORD ? ; 04

HYPERCUBE ENDS

.data

ALIGN WORD

CoordArray HYPERCUBE points DUP(<0,0,0>)

message BYTE "Please enter 5 coordinates: ", 0

questionX BYTE "Please enter an X value: ", 0

questionY BYTE "Please enter a Y value: ", 0

questionZ BYTE "Please enter a Z value: ",0

writeZ BYTE "The Z values are: ", 0

.code

main PROC

mov edx, OFFSET message

call WriteString

call crlf

mov edi, 0

mov ecx, points

L1:

mov eax, 0

mov edx, OFFSET questionX

call WriteString

call ReadInt

mov (HYPERCUBE PTR CoordArray[edi]).X, ax

mov edx, OFFSET questionY

call WriteString

call ReadInt

mov (HYPERCUBE PTR CoordArray[edi]).Y, ax

mov edx, OFFSET questionZ

call WriteString

call ReadInt

mov (HYPERCUBE PTR CoordArray[edi]).Z, ax

add edi, TYPE HYPERCUBE

loop L1

call crlf

mov edx, OFFSET writeZ

call WriteString

call crlf

mov ebx, OFFSET CoordArray

mov esi, 4

mov eax, 0

mov ecx, points

L2:

mov ax, [ebx+esi]

call WriteInt

call crlf

add ebx, SIZEOF HYPERCUBE

loop

exit

main ENDP

END main