**Question # 1**

Write down the instructions for given statements.

1. Generate and display ten pseudorandom signed integers in the range (0 – 99).

**Solution:**

mov ecx, 10

L1:

mov eax, 99

call RandomRange

loop L1

1. Pass each integer writeInt in eax and display it on a separate line.

**Solution:**

Write the following code below “call RandomRange” statement:

call WriteInt

call crlf

1. Input a string from the user:
   1. edx points to the string.

**Solution:**

mov edx, OFFSET userInput

* 1. ecx specifies the maximum number of characters the user is permitted to enter.

**Solution:**

mov ecx, LENGTHOF userInput

1. Display a null-terminated string and move the cursor to t he beginning of the next screen line (use embedded CR/LF).

**Solution:**

array BYTE “Jahanzeb”, 0dh, 0ah, 0 - .data section

mov edx, OFFSET array - .code section

call WriteString

# Question # 2

1. Write a program that copies signed numbers from one array to another but it copies numbers that are less than or equal to number entered by the user.

**Solution:**

Code:

include irvine32.inc

.data

array SDWORD -4,-10,-5,-9,-7,-2,-12,-5,-6,-13

array2 SDWORD 10 DUP (0)

msg BYTE "Enter negative number: ",0

msg2 BYTE "Numbers smaller or equal to input number: ",0

.code

main PROC

mov edx, OFFSET msg

call WriteString

mov eax, 0

call ReadInt

mov ebx, eax

call crlf

call crlf

mov ecx, LENGTHOF array

mov esi, OFFSET array

mov edi, OFFSET array2

L1:

mov eax, [esi]

cmp eax, ebx

JG above

mov [edi], eax

add edi, TYPE array2

above:

add esi, TYPE array

loop L1

mov edx, OFFSET msg2

call WriteString

call crlf

mov ecx, LENGTHOF array2

mov esi, OFFSET array2

L2:

mov eax, [esi]

cmp eax, 0

JE exitLoop

call WriteInt

call crlf

add esi, TYPE array2

loop L2

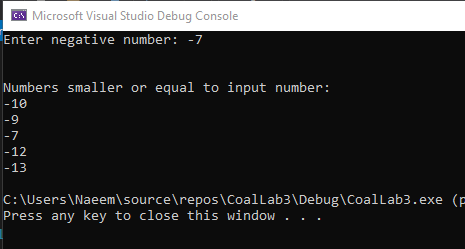
exitLoop:

exit

main ENDP

end main

Output:



1. Write a program get string from the user as character by character and store it in an array, then print the array.

Code:

include irvine32.inc

.data

msg BYTE "Enter a string: ",0

msg2 BYTE "Entered string: ",0

array BYTE 20 DUP (?)

.code

main PROC

mov edx, OFFSET msg

call WriteString

mov ecx, LENGTHOF array

mov esi, OFFSET array

L1:

call ReadChar

cmp al, 13

JE outofloop

call WriteChar

mov [esi], al

inc esi

loop L1

outofloop:

call crlf

call crlf

mov edx, OFFSET msg2

call WriteString

mov edx, OFFSET array

call WriteString

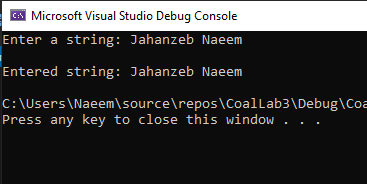
call crlf

exit

main ENDP

end main

Output:



# Question # 3

Write a procedure which is passed offset of an array of Dword and print each element of the array, on returning from the procedure, print the sum of the array.

**Solution:**

Code:

include irvine32.inc

.data

digits DWORD 5,3,8,5,0,2,9,6,7,1

msg BYTE "Elements of array: ", 0

msg2 BYTE "Sum of array elements: ", 0

.code

main PROC

mov ebx, OFFSET digits

mov ecx, LENGTHOF digits

push ebx

push ecx

call printAndSum

call crlf

call crlf

mov edx, OFFSET msg2

call WriteString

mov eax, ebx

call WriteDec

call crlf

exit

main ENDP

printAndSum PROC

mov edx, OFFSET msg

call WriteString

mov esi, [esp + 8]

mov ecx, [esp + 4]

mov ebx, 0

L1:

mov eax, [esi]

call WriteDec

add ebx, eax

add esi, 4

mov al, 32

call WriteChar

mov al, 44

call WriteChar

mov al, 32

call WriteChar

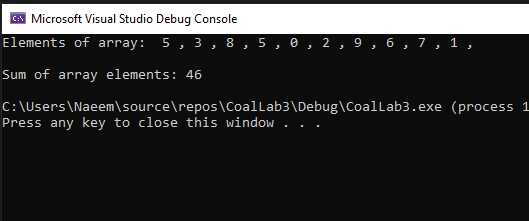
loop L1

ret

printAndSum ENDP

END main

Output:



# Question # 4

1. What are the three basic operations in stack? Explain with the help of an example.

**Solution:**

**Push:** this function adds a new value in stack and makes it the new top value of stack.

**Pop**: this function returns the top value in stack and then deletes it from the stack.

**Top**: this function also returns the top value in stack but it does not delete it from the stack.

**Example:** Suppose the Main procedure calls Procedure1 and then Procedure1 calls Procedure2. As the Main calls Procedure1, the return address of Main is pushed on to stack. And then, when Procedure1 calls Procedure2, the return address of Procedure1 is pushed on to the stack.

When Procedure2 ends, stack pops top value and control returns to Procedure1. And then, when Procedure1 ends, stack pops top value and control returns to Main procedure.

Another real world example is in web browsers. As the user visits different websites, address of each page is pushed on to stack. When user clicks back, value of top - 1 is returned and the browser goes back. And when the user closes the browser or clears history, all values in the stack are popped.

1. For stack implementation, Is given statement is true or not? Justify your answer

“Only words or double words could be saved into the stack, not a byte”.

**Solution:**

The statement is **true**.

The statement is true because in Irvine32, stack elements have 32-bit size and ESP is used to point to the top. We can also store 16-bit values in the stack using SS and SP. But, SP cannot be further divided into smaller parts.

BYTE values have 8-bit whereas the smallest value we can access from stack is 16 bit value (using SP). Hence, pushing 8-bit values in to the stack is not possible.