# Comp 3350: Computer Organization & Assembly Language

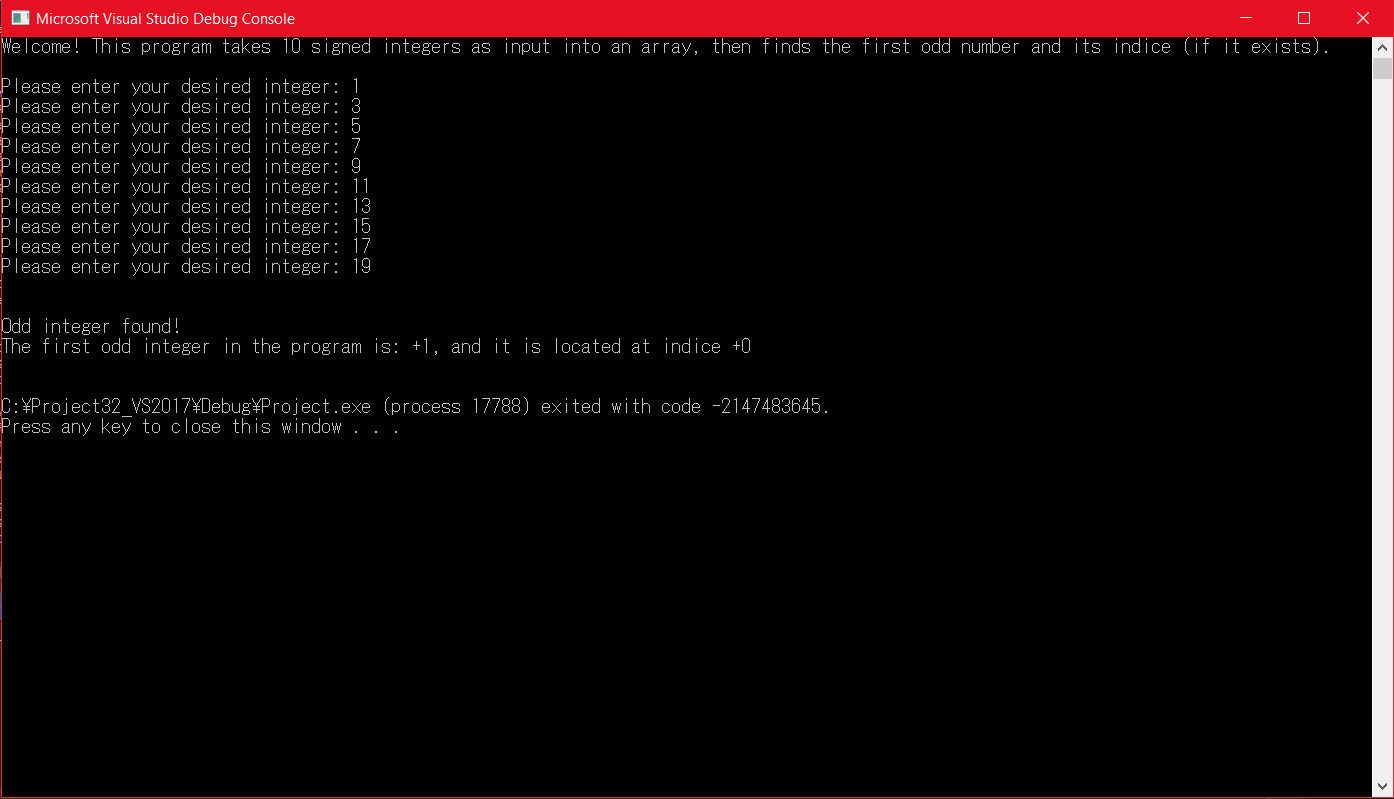
# HW # 7: Theme: Conditionals, Booleans, Loops

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

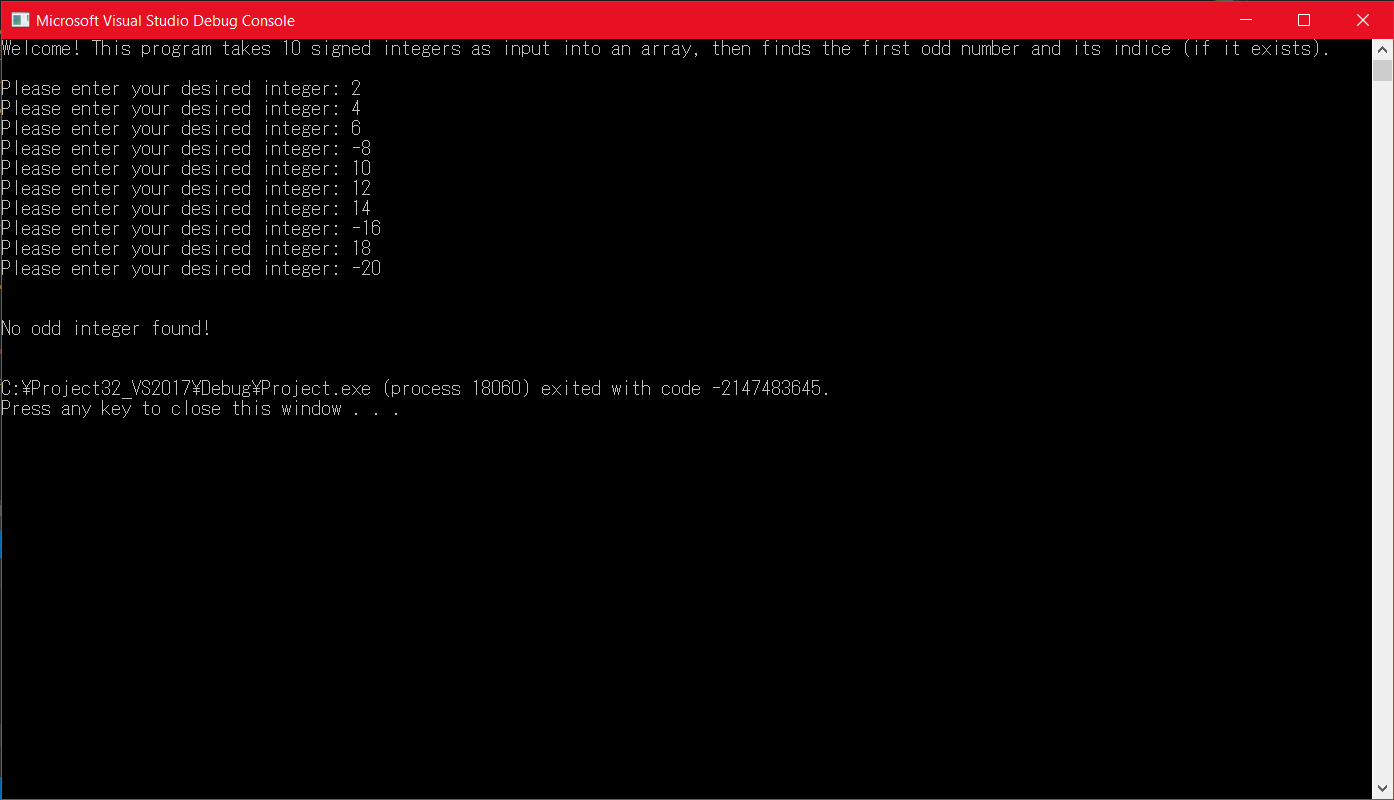
**Table of Contents:**

* Question 1 is answered on pages 1 – 2
* Question 2 is answered on page 3
* Question 3 is answered on pages 4 – 5
* Question 4 is answered on pages 6 – 7

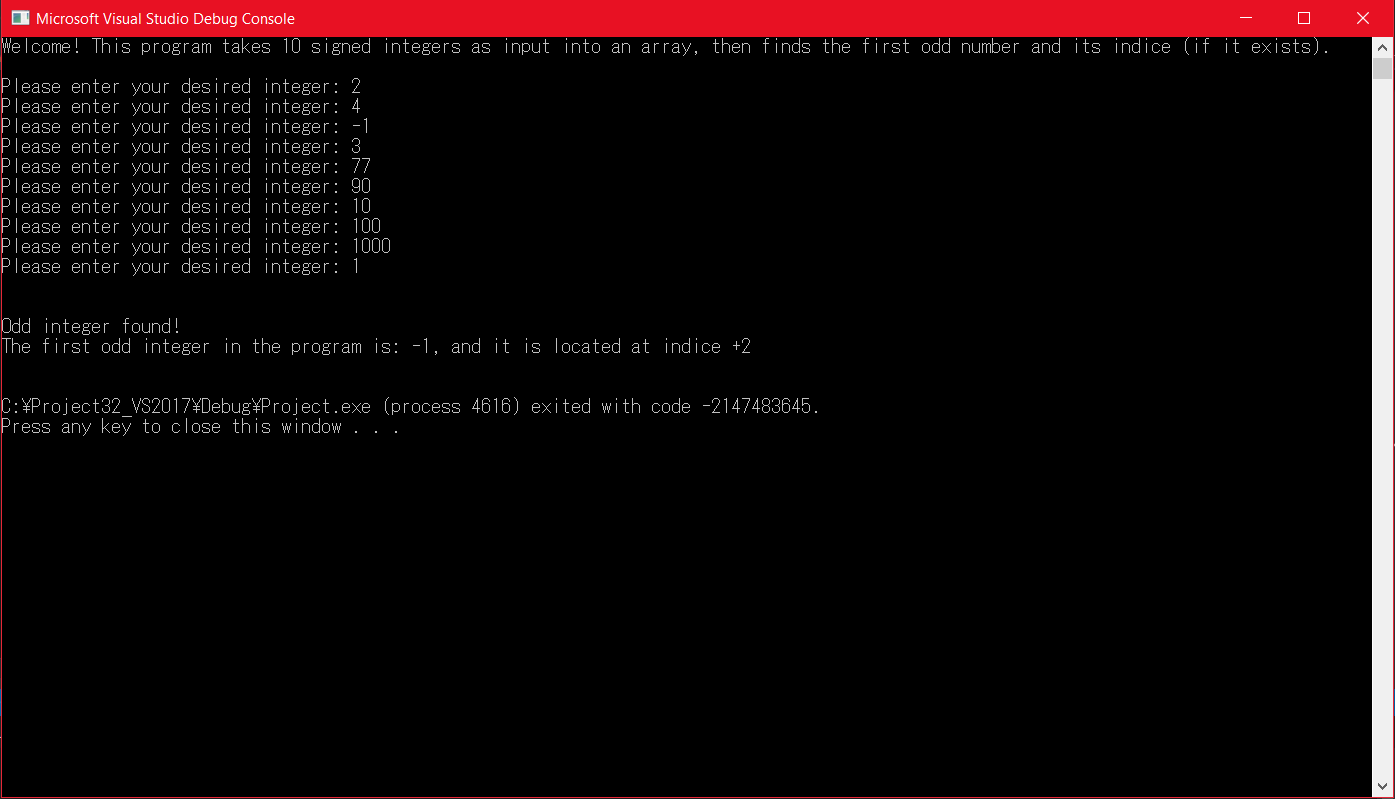
1. Draft a program that scans an array to determine the first odd integer in an array. If a value is found, the program should print “odd integer found” its value and index. If no odd integer is found, the program should print “no odd integer found.” Submit the list file and show runs for the following example arrays:
   1. Array has all odd integers

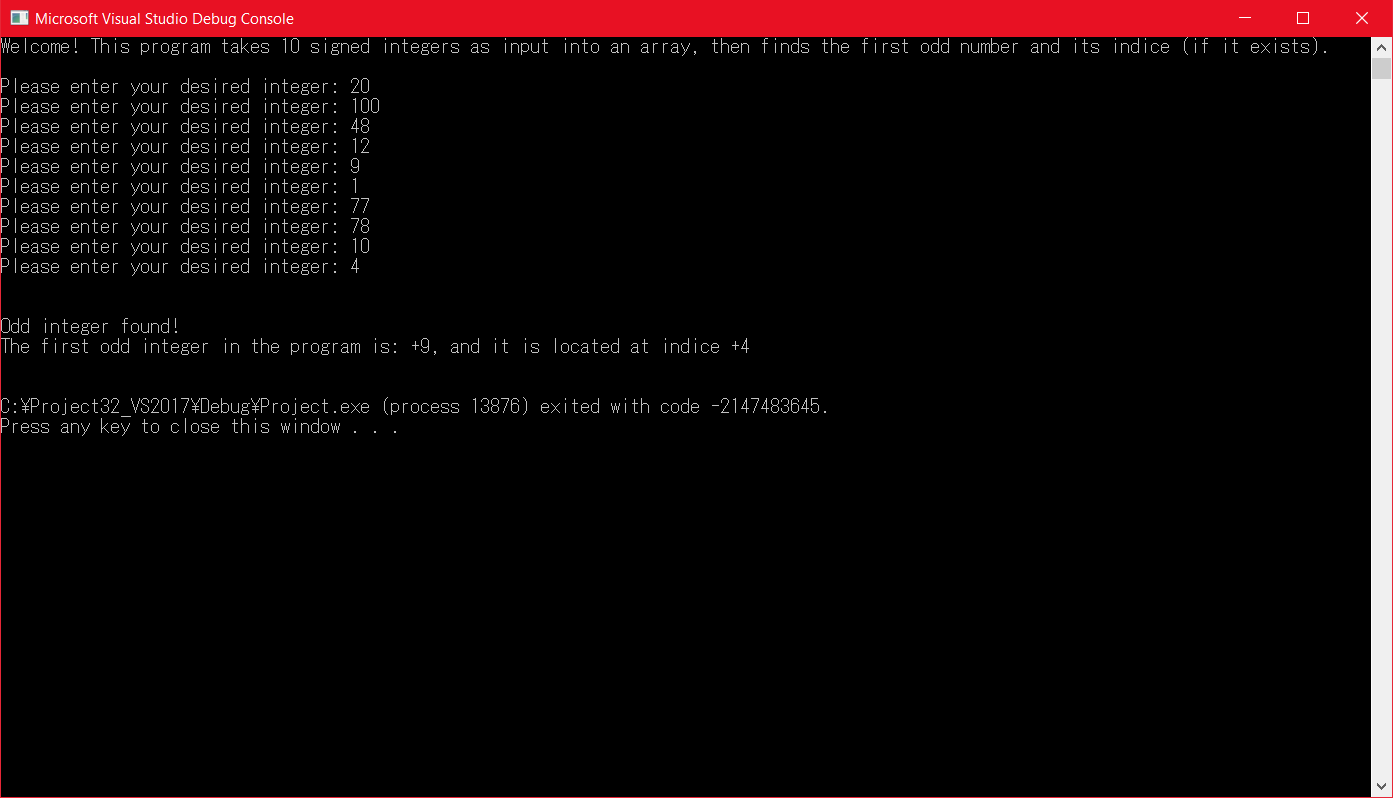


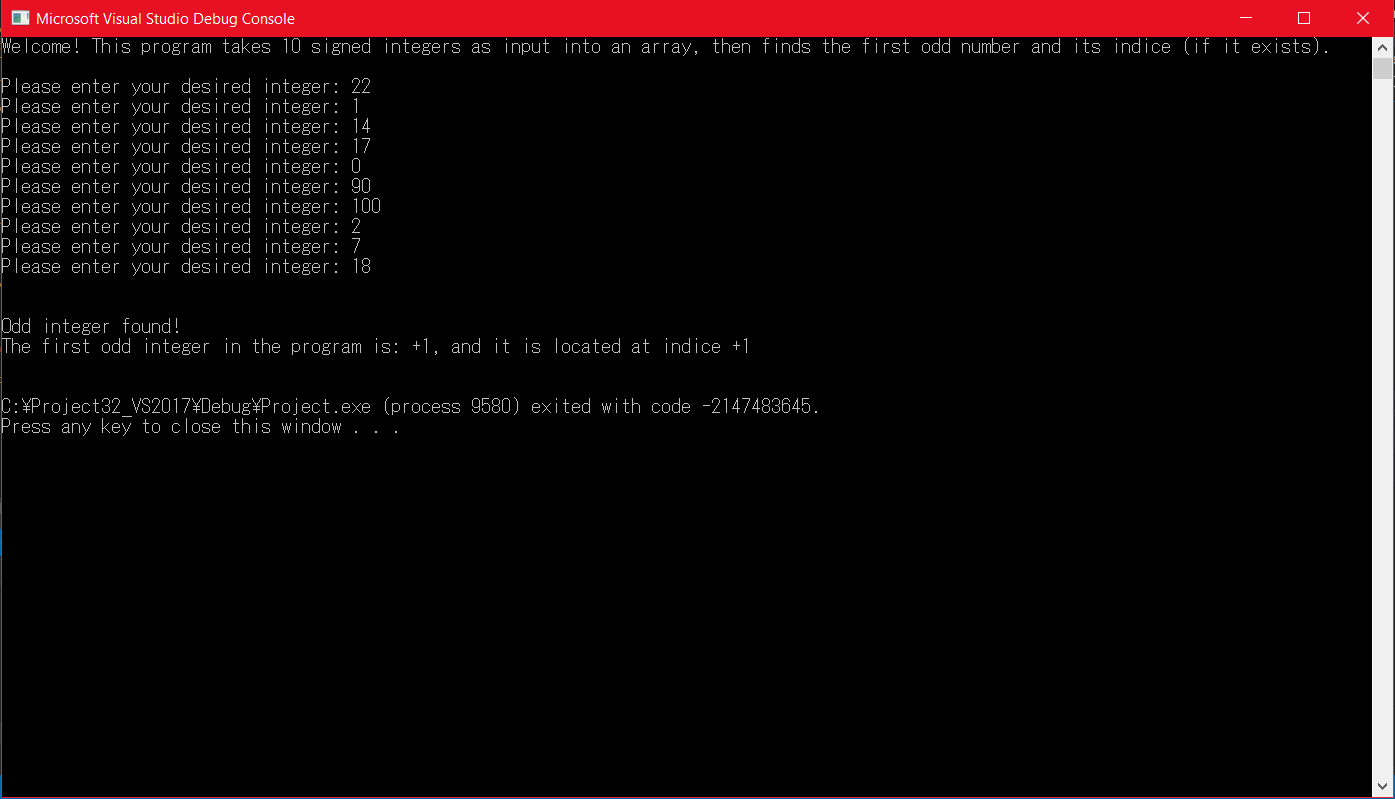
* 1. Array has all even integers



* 1. Several arrays with a mix of odd and even integers positioned at different indices

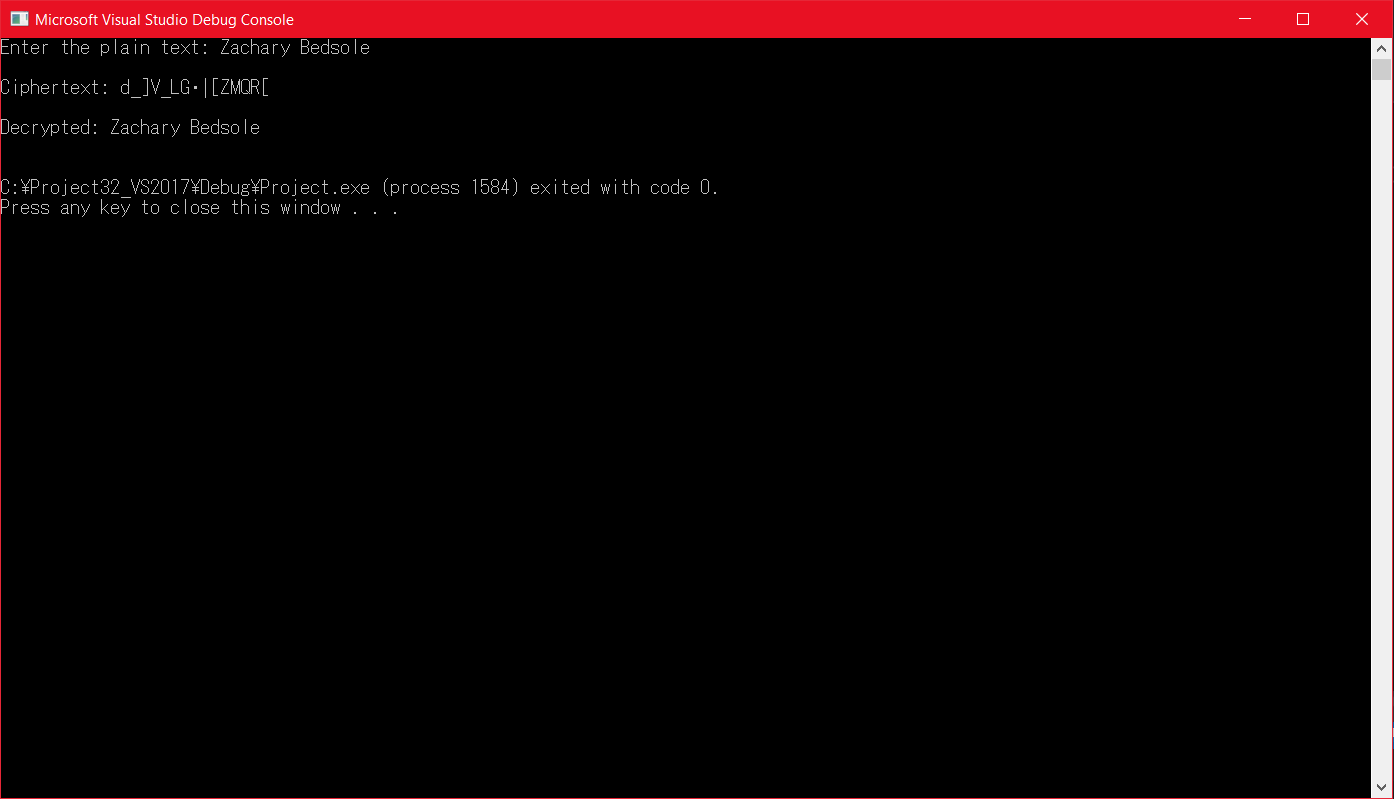






1. Write a program which encodes any string using the XOR instruction. Test it using your <first name last name> in the data segment to produce cipher text and then decode using the program to get plain text. Use the last three digits of your student id as the key. Print plane text from the data segment, print the cipher text, and then print the plain text upon execution. What are the strengths and weaknesses of this encryption method (25% of points)?

**Screenshot:**



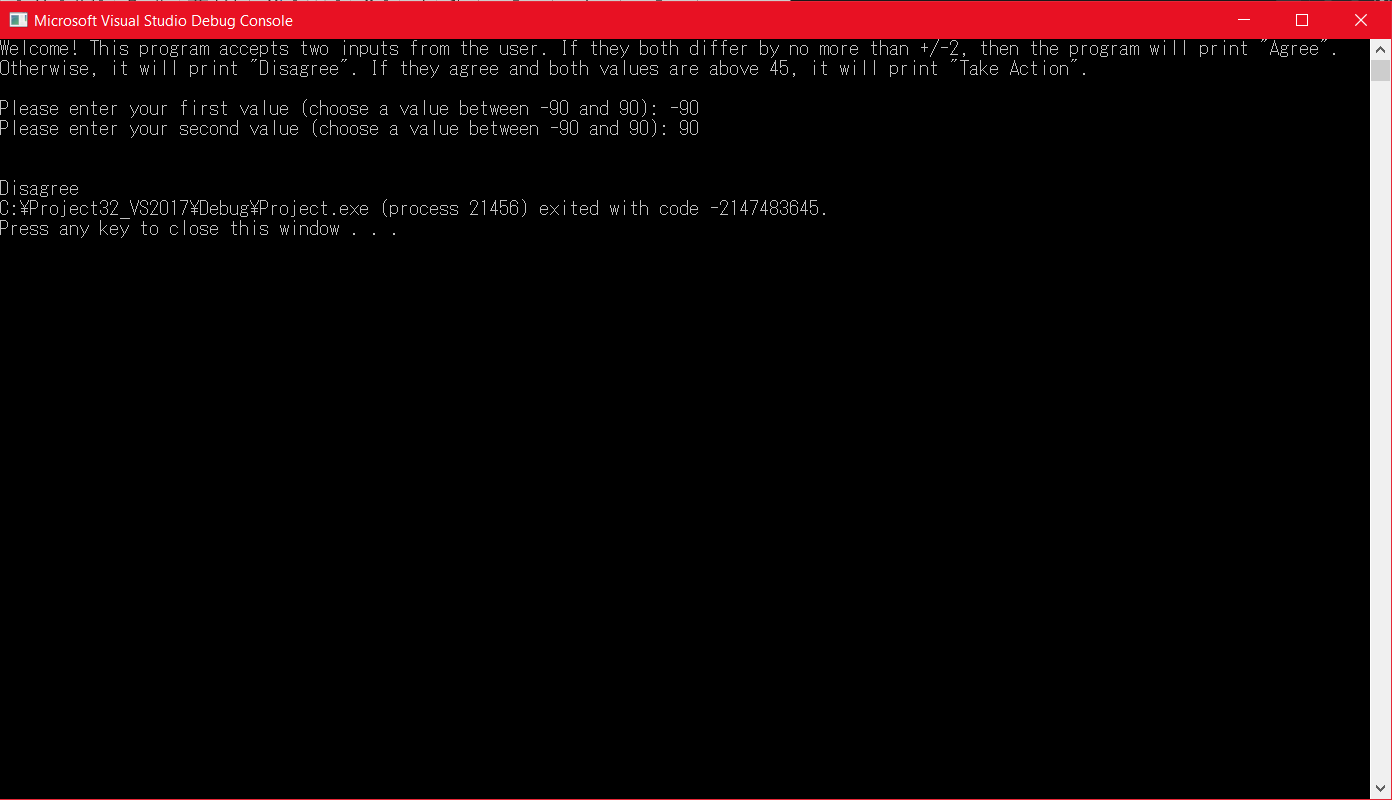
**Strengths and Weaknesses:** The strength of this encryption method is that it needs a special key to decrypt the encrypted message. Without this key, decrypting is impossible to do. This results in a type of encryption called *symmetric encryption*, where the key is used both for encrypting and decrypting. The encryption strength would be greatly improved if the key contained multiple characters, which leads us to the prime weaknesses within the encryption method. One such weakness to this encryption method is that the key can only be a value between 1 – 255, making a brute force attempt easy to do. All the brute force program would have to do is try each possible key for decrypting the string and display all possible results to the screen; it is likely that only one key would yield a logical plaintext, so the attacker could easily find the key and decrypt any string encrypted with it. Another key weakness is that this encryption method is subject to frequency analysis. For example, the letter e is in two places within my last name, and each encrypts to [, as opposed to one of them encrypting to something different and effectively “smearing out” the letters correspondents (the same could be said about the letter a encrypting to \_). If an attacker were to guess that [ decrypts to e (being that e is the most commonly used letter in the alphabet, and assuming the attacker tried to decrypt \_ to e first with no success) then they could effectively find out the key and decrypt the rest of the letters using said key. One other weakness is that the key would need to be privately exchanged between the sender and receiver, effectively making the encryption method prone to Man-in-the-Middle (MITM) attacks.

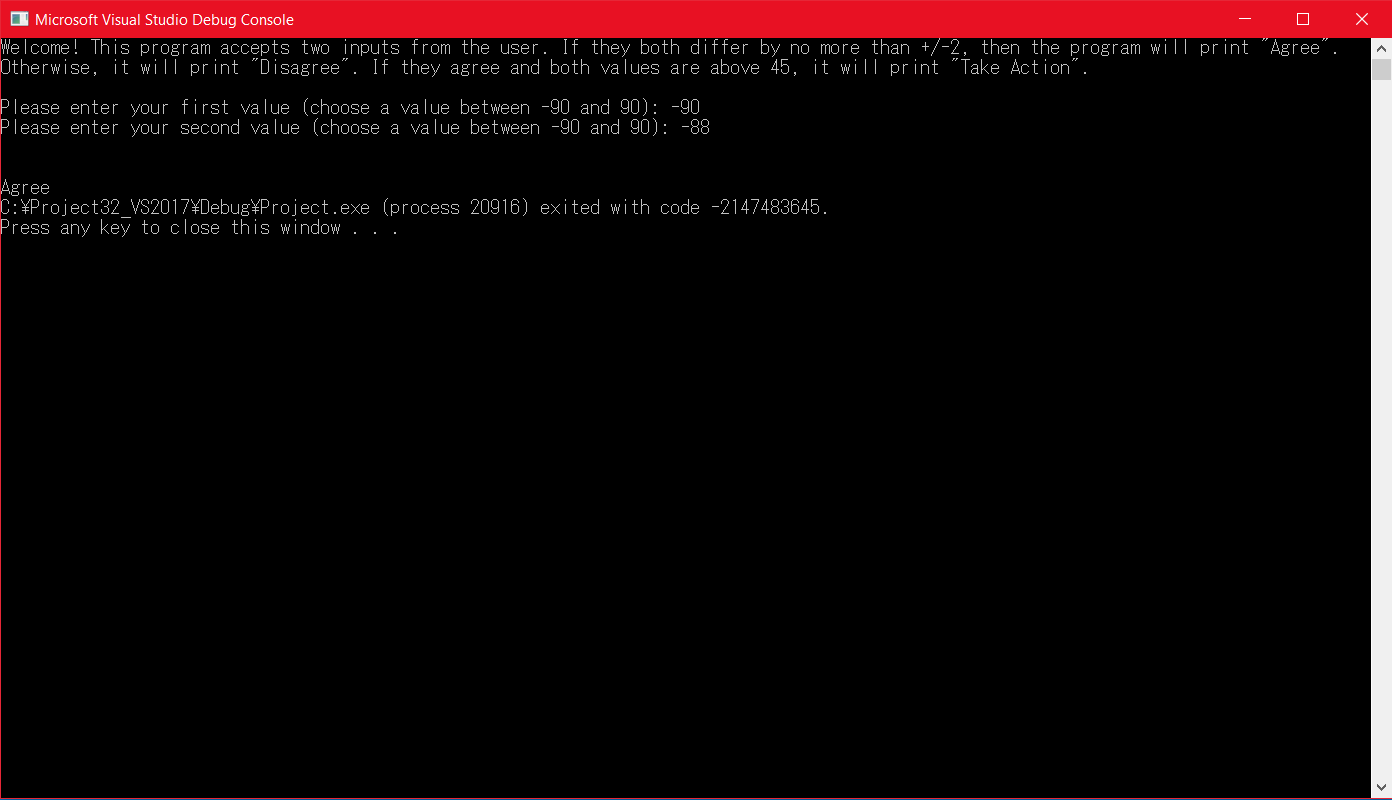
1. Write a program that gets its input from two sensors. If the values of the sensors differ by no more than +/- 2, print “Agree”, otherwise, print “Disagree.” You can assume that the values are integers. Additionally, if the values Agree and they are each more than 45, print “Take Action”. Submit list file and show robust testing for various inputs, including boundary conditions, in the closed interval (-90 … 90).

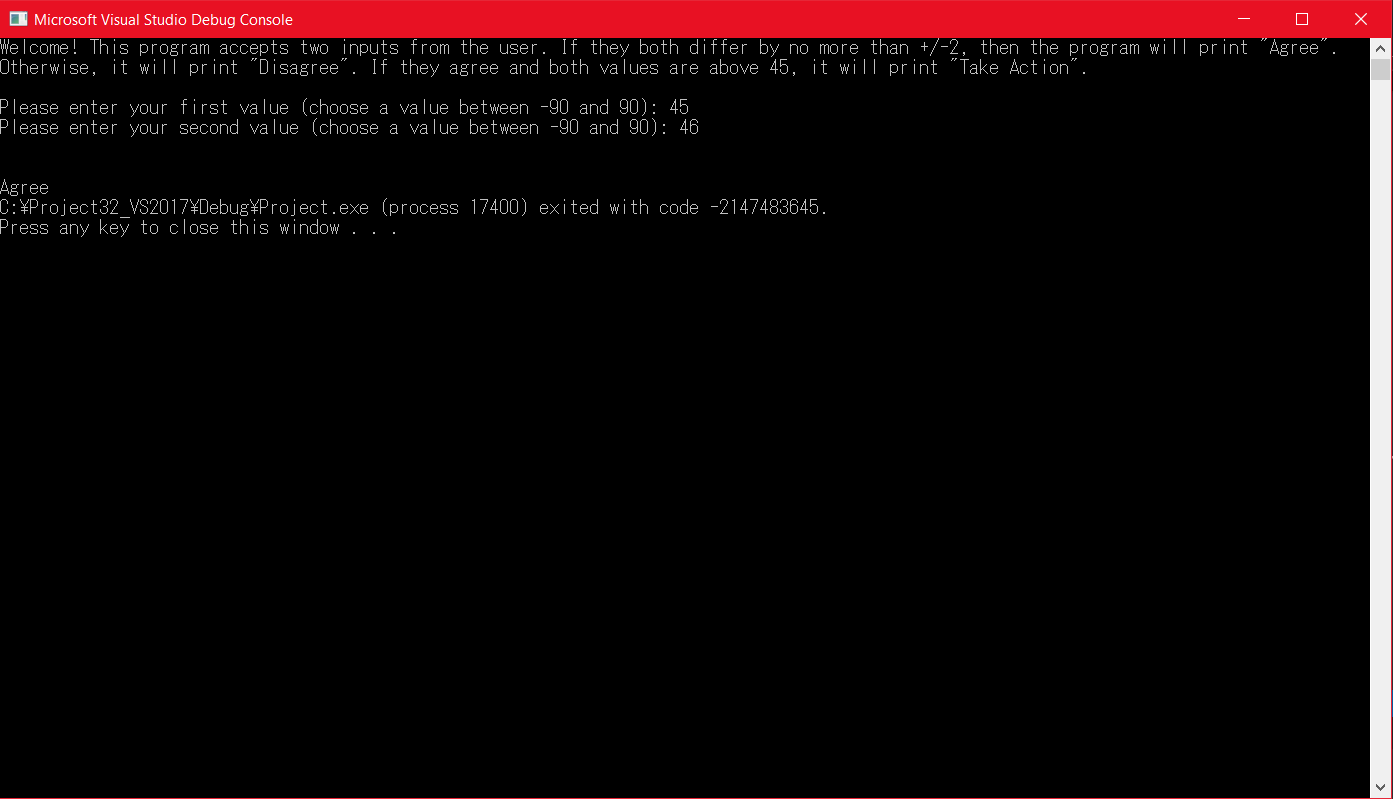
**Input test data**

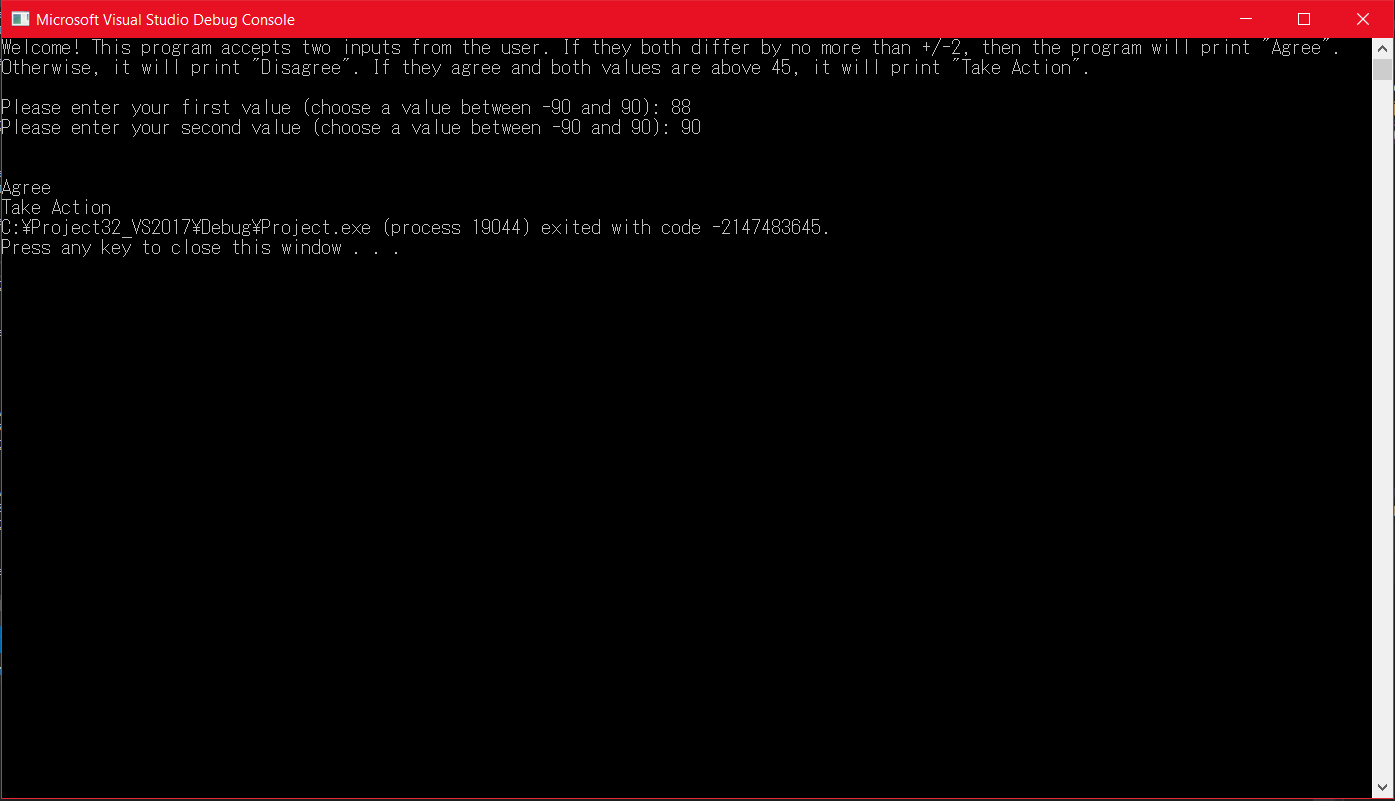
|  |  |  |
| --- | --- | --- |
| **CX** | **BX** | **Val1** |
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

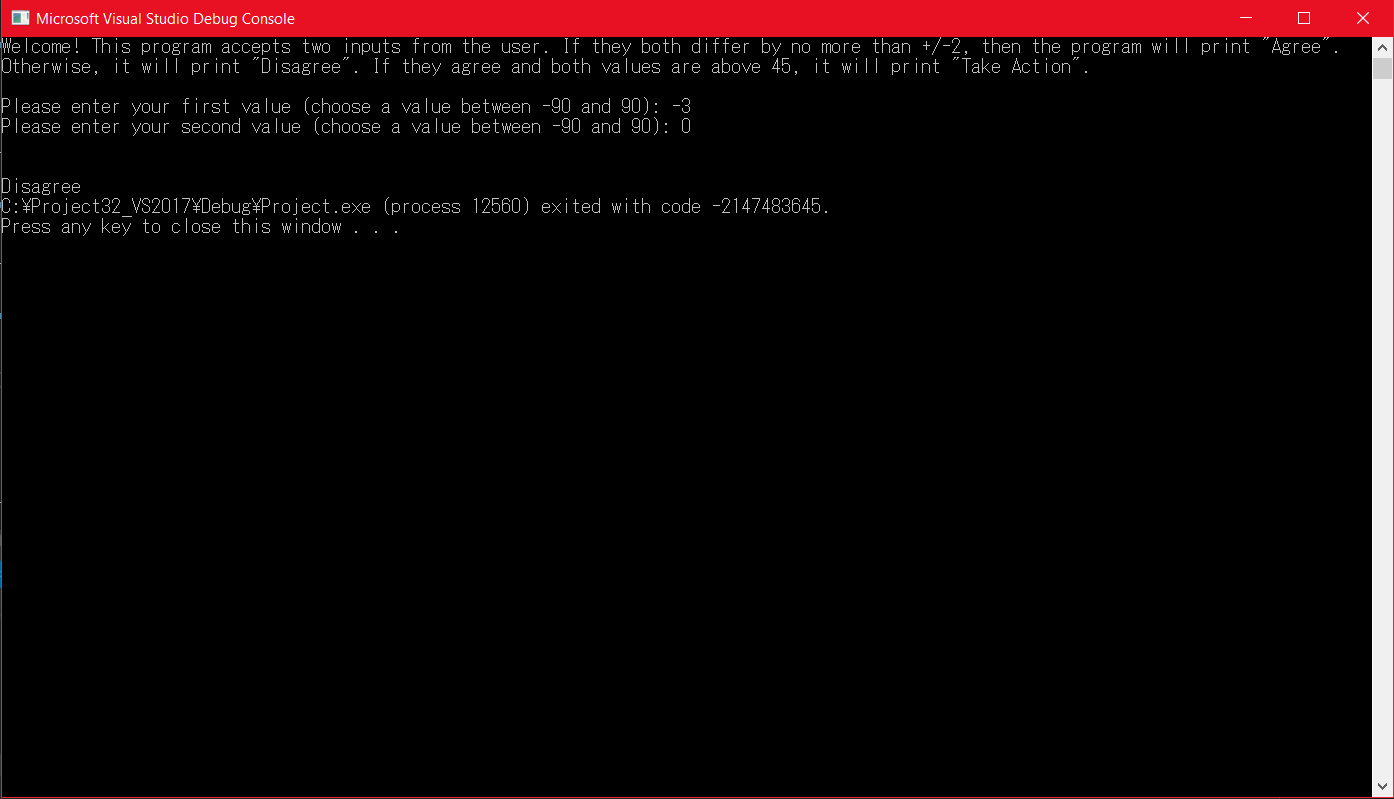
**Screenshots:**











1. Draw the stack (pencil-paper or word🡪pdf) at different points of the main and subroutine to show your understanding of the call and return functions.

One Proc

4040020 call FMul

4040026 mov eax, ebx

…

…

One EndP

FMul PROC

4041040 Push ecx

4041044 Push ebx

4041048 mov eax, edx

…

…

404A060 Pop ebx

404A062 Pop ecx

404A064 ret

FMul EndP

**Before Call to FMul PROC:**

Offset:

0000 1000 unknown (ESP)

0000 0FFC 0000 0000

0000 0FF8 0000 0000

0000 0FF4 0000 0000

EIP: 0404 0040

**After Call to FMul PROC:**

Offset:

0000 1000 unknown

0000 0FFC 0404 0046 (ESP)

0000 0FF8 0000 0000

0000 0FF4 0000 0000

EIP: 0404 1020

**After Pushing ECX:**

Offset:

0000 1000 unknown

0000 0FFC 0404 0046

0000 0FF8 ECX value (ESP)

0000 0FF4 0000 0000

**After Pushing EBX:**

Offset:

0000 1000 unknown

0000 0FFC 0404 0046

0000 0FF8 ECX value

0000 0FF4 EBX value (ESP)

**After Popping EBX:**

Offset:

0000 1000 unknown

0000 0FFC 0404 0046

0000 0FF8 ECX value (ESP)

0000 0FF4 0000 0000

**After Popping ECX:**

Offset:

0000 1000 unknown

0000 0FFC 0404 0046 (ESP)

0000 0FF8 0000 0000

0000 0FF4 0000 0000

**After Return to One:**

Offset:

0000 1000 unknown (ESP)

0000 0FFC 0000 0000

0000 0FF8 0000 0000

0000 0FF4 0000 0000

EIP: 0404 0046