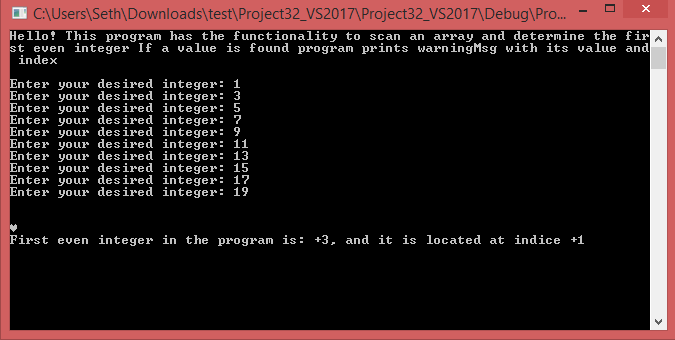
# Comp 3350: Computer Organization & Assembly Language

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

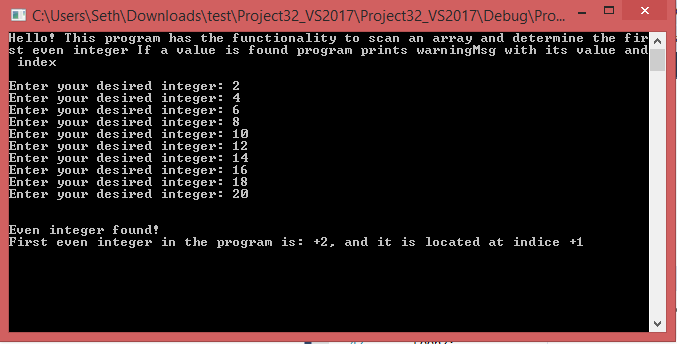
*(All main questions carry equal weight. Credit awarded to only those answers for which work has been shown.)*

1. Draft a program that scans an array to determine the first even integer in an array. If a value is found, the program should print “even integer found” its value and index. If no even integer is found, the program should print “no even integer found.” **Submit the asm/list file and screenshots that shows the output of your code for the following example arrays:** 
   1. Array has all odd integers
   2. Array has all even integers
   3. Several arrays with a mix of odd and even integers positioned at different indices

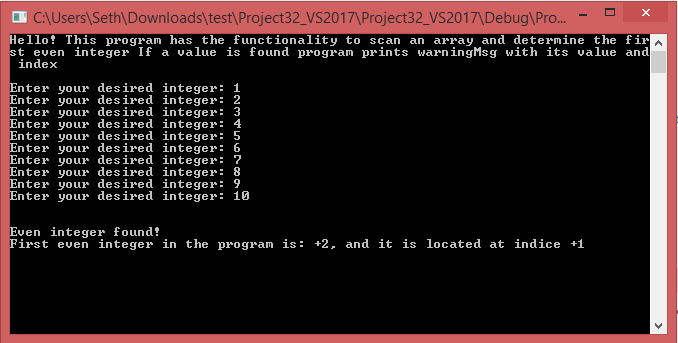
a:



b:

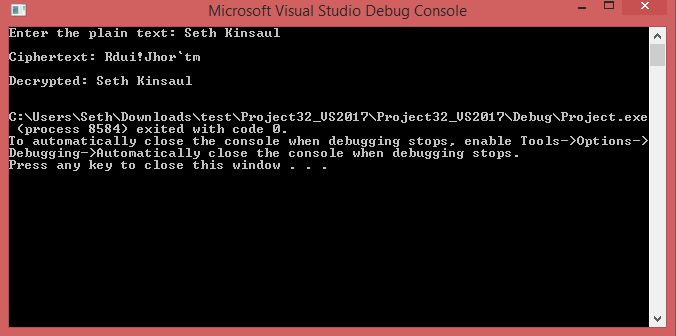


c:



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 two 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. **Submit the asm/list file and screenshots that shows the output of your code.**

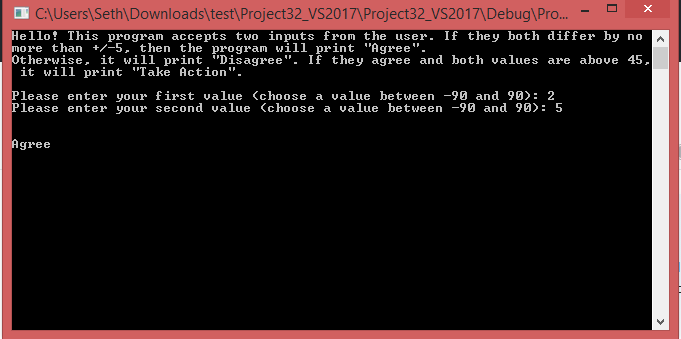
What are the strengths and weaknesses of this encryption method (**25% of points, Typewritten answer required**)?



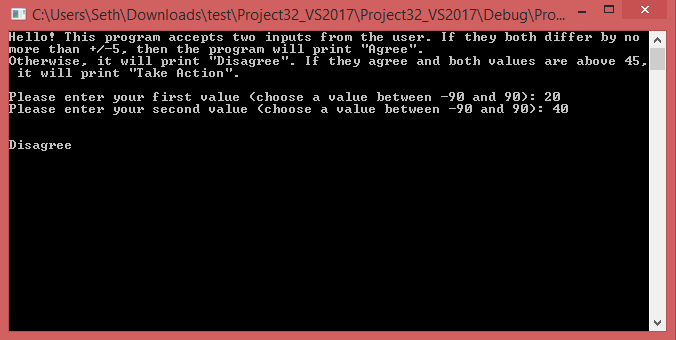
**Strengths and Weaknesses:** The strength of this encryption method is that it needs a special key to decrypt the encrypted message. This results in a type of encryption called *symmetric encryption*, where the key is used both for encrypting and decrypting. The encryption strength would be greatly improved if the key contained multiple characters, which leads us to the prime weaknesses within the encryption method. One weakness to this encryption method is that the key can only be a value between 1 – 255. This makes a brute force attempt easier 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 intruder 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 \_). The last weakness I can see is that the key would need to be privately exchanged between the sender and receiver. This 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 +/- 5, 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 asm/list file and show screenshots of robust testing for various inputs, including boundary conditions, in the closed interval (-90 … 90).**

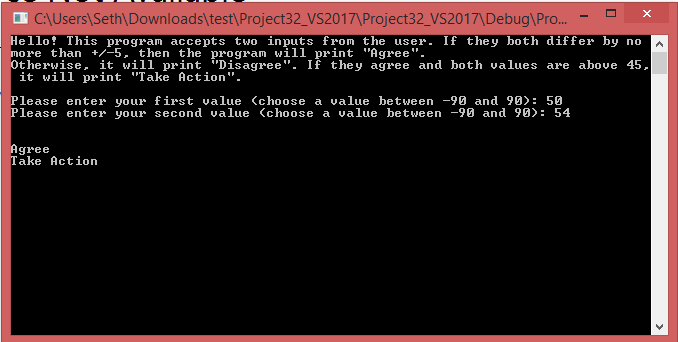
**Program Agrees (values differ by no more than +/-5):**



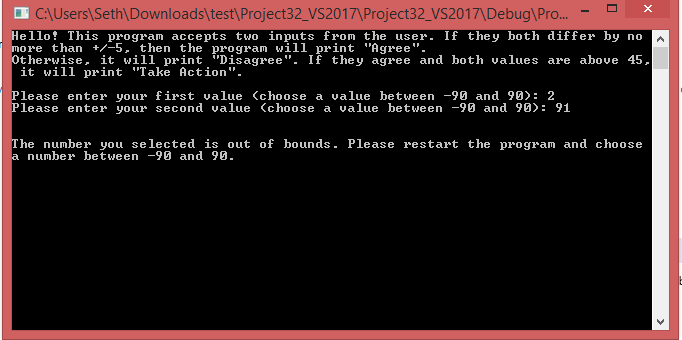
**Program Disagrees (value differ by more than +/-5):**



**Values are over 45 and agree:**



**One or more values are out of bounds:**



1. Draw the stack (word/pdf) before every instruction that is marked red is executed to show your understanding of the call and return functions. Use N/A to represent unpredictable values.

Main Proc

4040018 mov ecx, 0000000Ch

404001C mov ebx, 0000000Bh

4040020 call FMul

4040026 mov eax, ebx

…

…

Main EndP

FMul PROC

4041040 Push ecx

4041044 Push ebx

4041048 mov eax, edx

…

…

404A060 Pop ebx

404A062 Pop ecx

404A064 ret

FMul EndP

**First Step:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 0000

0000 0FF8 0000 0000

0000 0FF4 0000 0000

0000 0FF0 0000 0000

EIP: 0404 0018

**Second Step:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B

0000 0FF8 0000 0000

0000 0FF4 0000 0000

0000 0FF0 0000 0000

EIP: 0404 001C

**Before Call to FMul PROC:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B

0000 0FF8 0000 0000

0000 0FF4 0000 0000

0000 0FF0 0000 0000

EIP: 0404 0020

**After Call to FMul PROC:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B

0000 0FF8 0404 1040 (ESP)

0000 0FF4 0000 0000

0000 0FF0 0000 0000

EIP: 0404 1040

**After Pushing ECX:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B

0000 0FF8 0404 1040

0000 0FF4 ecx value (ESP)

0000 0FF0 0000 0000

**After Pushing EBX:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B

0000 0FF8 0404 1040

0000 0FF4 ecx value

0000 0FF0 ebx value (ESP)

**After Popping EBX:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B

0000 0FF8 0404 1040

0000 0FF4 ecx value (ESP)

0000 0FF0 0000 0000

**After Popping ECX:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B

0000 0FF8 0404 1040 (ESP)

0000 0FF4 0000 0000

0000 0FF0 0000 0000

**After Return:**

Offset:

0000 1000 0000 000C

0000 0FFC 0000 000B (ESP)

0000 0FF8 0000 0000

0000 0FF4 0000 0000

0000 0FF0 0000 0000

EIP: 0404 0046