CSCI2160Proj6F19Recursion&Encryption **Project – Due: 12/7/2019, 9:00 AM**

**NOTE: If you hope to be excused from the final exam, your project 6 will have to be submitted by Thursday, Dec. 5, 2019 by 11:59 AM. That will give us time to get it graded to let you know if you will be exempt from the final.**

Recursion - **proj6a.asm (15 pts)**

Write the assembly code for the method where **fun** is defined as below.

1, n=0  
 fun(n) = 2, n=1  
 7, n=2  
 fun(n-1) + 2\*fun(n-2)+3\*fun(n-3), for n>2

Write your own driver **proj6b.asm** to test your external methods. For this assignment after you have dropped every .asm file necessary to assemble **proj6b.asm**  save every file again as a **.txt**  file and upload all of the **.txt** files to D2L as well. The file must have a .txt extension so it can be run through **TurnItIn**.

Create a file called **convertMethods.asm** that will contain all of the below methods. Write them to be “invoked”.

1. (20 pts) **+hexToCharacter(lpDestination:dword, lpSource:dword, numBytes:dword):void**

This method accepts 3 parameters.

1. The first parameter is the address of the string that will contain the hex representation of the characters which will then be able to display the characters in “hex”.
2. The 2nd parameter is to be interpreted as two possible things depending on the value of the 3rd.
   1. the address of a string of bytes containing binary values that will be converted to their equivalent ASCII values and stored at lpDestination (if the 3rd parameter >0 ).
   2. Simply a dword whose contents are to be converted to ASCII (if the 3rd parameter is 0)
3. the 3nd parameter will be one of two possible things:
4. the value **0** which will mean that the 2nd parameter is a 4-byte dword which will be converted to an equivalent ASCII string of 8 bytes, null-terminated and stored at lpDestination. For example, if EBX contains 1234ABCDh then the string of ASCII bytes representing those hex digits would be 313233344142434400 (which when used with putstring, would display 1234ABCD on the screen.
5. The number of bytes (numBytes) to be converted is greater than 0. This means, then, the 2nd parameter is the address of a string containing at least numBytes which will be converted to its equivalent null-terminated printable ASCII string of characters and stored starting at the address stored in lpDestination.. For example the string to convert might be something like “John went to town riding on a pony”,0 in which case the lpDestination string would contain the hex characterization for numBytes of that string. There are 34 characters in this particular string so the method would build a string of 68 ASCII values where each byte is the ASCII representation for one character and the 69th byte would be all 00h.

This method would take the string strChar (from below example) of hex bytes: 4142434445464700 and create the string of characters: 3431343234333434343534363437303000 which , if displayed later, would display the characters   
**4 1 4 2 4 3 4 4 4 5 4 6 4 7 0 0**

strChar byte “ABCDEFG”,0  
strHexChars 80 dup(?) ;holds converted string of characters

puts strChar ;displays ABCDEFG  
**INVOKE hexToCharacter,ADDR strHexChars, ADDR strChar, 8** ;convert hex digits to  
 ; equivalent ASCII  
**INVOKE putstring,ADDR strHexChars ;displays 4 1 4 2 4 3 4 4 4 5 4 6 4 7 0 0**

**IMPORTANT**

**If you are unable to get your hexToCharacter method working correctly, you can use MY equivalent method hexToChar which will do the same thing as yours.**

2. (25 pts) Write the method

**+charTo4HexDigits(lpSourceString:dword):dword**

which accepts a null-terminated string of characters and returns the dword mask derived from those characters. For example, suppose the user is prompted to enter an 8-hex digit mask and then enters ABCDEF01. This is accepted as an ASCII string. This method converts that string to the dword ABCDEF01h. The user can enter uppercase or lowercase A-F, a-f. If there is any invalid character in the string(not in the set {ABCDEF0123456789} or the set {abcdef0123456789} or the string is more than 8 characters long, the method returns -1.

3)( 25 pts) Write the method: **+encrypt32Bit(lpSourceString:dword,dMask:dword,numBytes:dword):dword**

Which will encrypt a null-terminated string of characters using a 32-bit mask key. This means that 4 bytes at a time will be encrypted. The first parameter is the address of the string to be encrypted. The second parameter is the dword mask to be used for encryption. The 3rd parameter is the number of bytes to be encrypted. The method will dynamically allocate (for this assignment a maximum of 256 bytes) to hold the encrypted string. Its address is returned to the calling program.

With encryption, trying to display the encrypted string as a string of displayable characters may not produce a meaningful display or even printable characters. That’s why you will use the **hexToCharacter** method to construct a “meaningful” display of the encrypted values. Your driver will input a source string from the keyboard (of max size 100 chars). Then, it will input the 32-bit encryption key. Display the hex representation of both the source string and then the encryption key with appropriate descriptions. Then, encrypt the source string. Display the original string and the encrypted string so the bytes of the strings line up. Then re-encrypt the newly encrypted string and you should get the original string. NOTE: If you write your encryption method incorrectly you may see the encryption and de-cryption work “correctly.” However, it is possible that you did NOT encrypt correctly. Thus, I will use my own encryption method to de-crypt your encrypted string. It should produce the exact original string.

**IMPORTANT!! You MUST use shifting operations and you MUST use the XOR method across all 4 bytes. You can apply your Xor’ing. I wanted an exercise that would allow you to get some experience using the shift instructions without contriving some weird example. There is a fairly simple method in which you could XOR a byte at a time, but you MUST XOR in groups of 4 bytes except for any “left-over” bytes. if the string has a length that is not a multiple of 4, then use the left-most bytes of the mask. for example if the mask is ABCD0123h then you would mask 4 bytes at a time from the string. Suppose that there is 1 extra byte beyond a multiple of 4, then you would mask that one extra byte with ABh, if there were 2 bytes left over, mask them with ABCD, etc.**

4) (15 pts) Write the driver to test your methods.