```
// The text encryption program in C++ and ASM with a very simple example encryption method - it simply adds 1 to the character.
// The encryption method is written in ASM. You will replace this with your allocated version for the assignment.
// In this version parameters are passed via registers (see 'encrypt' for details).
//
// Author: A.Oram (Feb 2018)
// Last revised Jan 2021 by A.Hamilton

char EKey = 'h';               // Replace x with your Encryption key

#define StudentName "Kemas Raihan"

//  *** PLEASE CHANGE THE NAME IN QUOTES ABOVE TO YOUR NAME  ***
//  *** KEEP ALL COMMENTS UP-TO-DATE. COMMENT USEFULLY ALL CODE YOU PRODUCE. DELETE STALE COMMENTS (LIKE THIS ONE) ***

#define MAXCHARS 6             // feel free to alter this, but 6 is the minimum

#include <string>              // for strings
#include <fstream>             // file I/O
#include <iostream>            // for cin >> and cout <<
#include <iomanip>             // for fancy output
#include "TimeUtils.h"         // for GetTime, GetDate, etc.

#define dollarchar '$'         // string terminator

char OChars[MAXCHARS],         // Original character string
   EChars[MAXCHARS],           // Encrypted character string
   DChars[MAXCHARS] = "Soon!"; // Decrypted character string

//---------------------------- C++ Functions ------------------------------------------------------

void get_char(char& a_character) // char& means passing the address of a_character
{
  a_character = (char)_getwche();

  if (a_character == '\r' || a_character == '\n')  // allow the enter key to work as the terminating character too
  {
    a_character = dollarchar;
  }
  __asm
  {
      OR dword ptr[eax], 20h                    // convert any uppercase letters to lowercase
  }
}

void get_original_chars(int& length)
{
  char next_char = ' ';
  length = 0;
  get_char(next_char);

  while ((length < MAXCHARS) && (next_char != dollarchar))
  {
    OChars[length++] = next_char;
    get_char(next_char);
  }
}

//-------------------------------------------------------------------------------------------------------------
//----------------- ENCRYPTION ROUTINES ---------------------------------------------------------------------

void encrypt_chars (int length, char EKey)
{
  char temp_char;                       // Character temporary store

  for(int i = 0; i < length; ++i)      // Encrypt characters one at a time
  {
    temp_char = OChars[i];              // Get the next char from Original Chars array
                                        // Note the lamentable lack of comments below!
    __asm
    {
      push   eax                        // save value in EAX register to the stack
      push   ecx                        // save value in ECX register to the stack
      push   edx                        // save value in EDX register to the stack

      movzx  ecx, temp_char             // use ECX register to store the ASCII value of temp_char for the encryption routine
      lea    eax, EKey                  // use EAX register to store the effective address of Ekey for the encryption routine

      push ecx                          // pass the first parameter to encrypt_3 for stdcall subroutine
      push eax                          // pass the second parameter to encrypt_3 for stdcall subroutine
      call   encrypt_3                  // perform encryption
      mov    temp_char, dl              // replace original value with new encrypted value

      pop    edx                        // restore original EDX value from stack
      pop    ecx                        // restore original ECX value from stack
      pop    eax                        // restore original EAX value from stack
    }

    EChars[i] = temp_char;              // Store encrypted char in the Encrypted Chars array
  }

  return;

  // Encrypt subroutine. You should paste in the encryption routine you've been allocated from BB and
  // overwrite this initial, simple, version. Ensure you change the #call# above to use the
  // correct 'encrypt_nn' label where nn is your encryption routine number.

  // Inputs: register EAX = 32-bit address of Ekey,
  //                  ECX = the character to be encrypted (in the low 8-bit field, CL).

  // Output: register EDX = the encrypted value of the source character (in the low 8-bit field, DL).
  // REMEMBER TO UPDATE THESE COMMENTS AS YOU DO THE ASSIGNMENT. DELETE OLD/STALE COMMENTS.

  __asm
  {
  encrypt_3: // Original encryption subroutine:
```

```
            //push  edx                    // EDX register will be used for encryption of the temp_char value, so save a copy of the original 
value
            //push  ecx                    // save a copy of the value of temp_char on the stack for later use
            //push  eax                    // EAX register will be used for encryption of the temp_char value, so save a copy of the effective 
address of Ekey
            //movzx eax, byte ptr[eax]     // retrieve Ekey value for encryption
            //rol   al, 1                  // encrypt the Ekey value by rotation of its bits to the left
            //not al                       // further encrypt the Ekey value by inversion of its bits
            //rol   al, 1                  // further encrypt the Ekey value by rotation to the left again
            //rol   al, 1                  // repeat previous instruction
            //mov   edx, eax               // the EAX register will need to restore effective address of EKey for the next couple of 
instructions
            //pop   eax                    // clean up the stack and also for next instruction
            //mov   byte ptr[eax], dl      // replace with new encrypted Ekey value for next character to be encrypted
            //pop   ecx                    // we shall encrypt the temp_char value for the next few steps and to also clean up the stack
            //xor ecx, edx                 // encrypt the temp_char value by implementing EXCLUSIVE OR logical disjunction with encrypted Ekey 
value
            //mov   eax, ecx                              // use EAX for further encryption of temp_char
            //ror   al, 1                                 // further encrypt the temp_char value by rotation of bits to the right
            //ror   al, 1                                         // repeat previous instruction
            //ror   al, 1                                         // repeat previous instruction
            //pop   edx                                           // clean up the stack
            //mov   edx, eax                              // use EDX as return value
            //ret                                                     // return to call site

            // Encryption subroutine using stdcall:
            push ebp                       // save call site base pointer on the stack
            mov ebp, esp                   // create a new stack frame for stdcall subroutine

            mov ebx, [ebp + 08h]           // we shall retrieve the ASCII value of Ekey by using the EBX register to point at its address in 
the stack
            movzx eax, byte ptr[ebx]       // retrieve the Ekey value for encryption
            rol   al, 1                    // encrypt the Ekey value by rotation of its bits to the left
            not al                         // further encrypt the Ekey value by inversion of its bits
            rol   al, 2                    // further encrypt the Ekey value by rotation to the left again
            mov   byte ptr[ebx], al        // replace with new encrypted Ekey value for next character to be encrypted
            mov   ecx, [ebp + 0Ch]         // restore temp_char for encryption
            xor ecx, eax                   // encrypt the temp_char value by implementing EXCLUSIVE OR logical disjunction with encrypted Ekey 
value
            mov   eax, ecx                                // use EAX for further encryption of temp_char
            ror   al, 3                                           // further encrypt the temp_char value by rotation of its bits to the right
            mov   edx, eax                                // use EDX as return value

            mov esp, ebp                   // make sure stack pointer is pointing at base pointer
            pop ebp                        // stdcall routine has completed so restore call site's base pointer
            ret 8                                                   // return to call site and clean up the stack
  }

  //--- End of Assembly code
}
//*** end of encrypt_chars function
//---------------------------------------------------------------------------------------------------------------




//---------------------------------------------------------------------------------------------------------------
//----------------- DECRYPTION ROUTINES -------------------------------------------------------------------------
//
void decrypt_chars (int length, char EKey)
{
  /*** To be written by you ***/

  return;
}

//*** end of decrypt_chars function
//---------------------------------------------------------------------------------------------------------------





//************ MAIN PROGRAM *************************************************************************************

int main(void)
{
    int i = 0;
    int char_count(0);  // The number of actual characters entered (upto MAXCHARS limit).

    std::cout << "\nPlease enter upto " << MAXCHARS << " alphanumeric characters:  ";

    get_original_chars(char_count);        // Input the original character string to be encrypted


    //*****************************************************
    // Open a file to store results (you can view and edit this file in Visual Studio)

    std::ofstream EDump;
    EDump.open("EncryptDump.txt", std::ios::app);
    EDump << "\n\nFoMCA Encryption program results (" << StudentName << ") Encryption key = '" << EKey << "'";
    EDump << "\nDate: " << GetDate() << "  Time: " << GetTime();


    //*****************************************************
    // Display and save to the EDump file the string just input

    std::cout << "\n\nOriginal string =  " << OChars << "\tHex = ";
    EDump << "\n\nOriginal string =  " << OChars << "\tHex = ";

    for (int i = 0; i < char_count; ++i)
    {
        std::cout << std::hex << std::setw(2) << std::setfill('0') << ((int(OChars[i])) & 0xFF) << "  ";
        EDump << std::hex << std::setw(2) << std::setfill('0') << ((int(OChars[i])) & 0xFF) << "  ";
    }
```

```
    //*****************************************************
    // Encrypt the string and display/save the result

    encrypt_chars(char_count, EKey);

    std::cout << "\n\nEncrypted string = " << EChars << "\tHex = ";
    EDump << "\n\nEncrypted string = " << EChars << "\tHex = ";
    for (int i = 0; i < char_count; ++i)
    {
        std::cout << ((int(EChars[i])) & 0xFF) << "  ";
        EDump << ((int(EChars[i])) & 0xFF) << "  ";
    }


    //*****************************************************
    // Decrypt the encrypted string and display/save the result

    decrypt_chars(char_count, EKey);        //**** YOU NEED TO WRITE THE BODY OF THIS FUNCTION ***

    std::cout << "\n\nDecrypted string = " << DChars << "\tHex = ";
    EDump << "\n\nDecrypted string = " << DChars << "\tHex = ";
    for (int i = 0; i < char_count; ++i)
    {
        std::cout << ((int(DChars[i])) & 0xFF) << "  ";
        EDump << ((int(DChars[i])) & 0xFF) << "  ";
    }
    //*****************************************************


    std::cout << "\n\n\n";
    EDump << "\n\n-----------------------------------------------------------------";
    EDump.close();

    system("PAUSE"); // do not use in your programs! just a hack to pause the program before exiting

  return 0;
} // end of whole encryption/decryption program ----------------------------------------------------------------------
```