Quinn Roemer

CISP - 440

Assignment 9.42

11/29/2018

Part 0 - Decryption Implementation

Description:

The goal of this assignment was to implement some code that decrypts characters. We were provided code that was already a fully functioning encryption algorithm. However, our task was to reverse that algorithm to allow it to decrypt itself.

Note: In my testing I used a text file read as binary numbers. This was then encrypted and decrypted into different text files to show the process. In my output I will show the original, encrypted, and decrypted files.

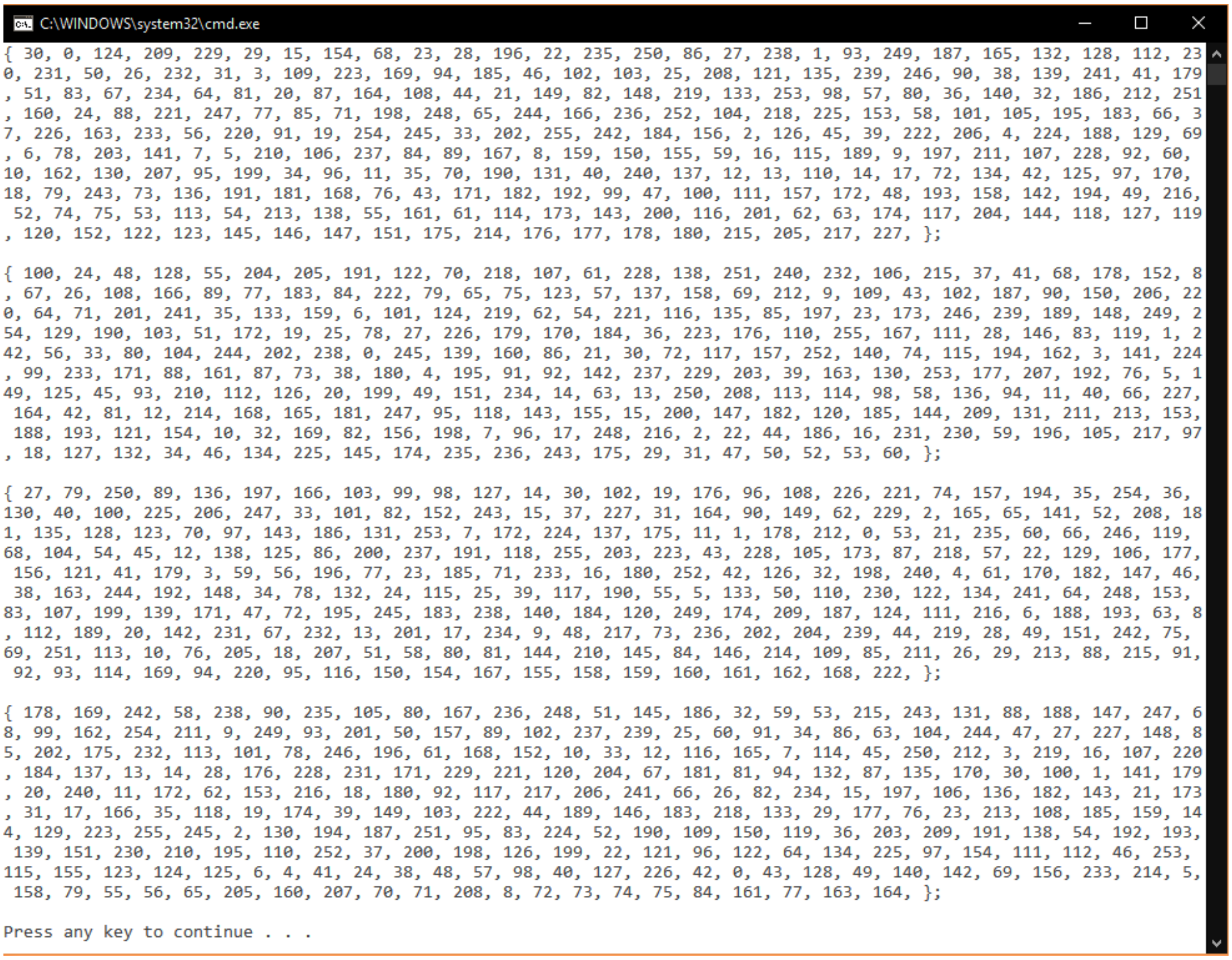
Source Code:

```
//Code written by Quinn Roemer 11/27/2018. Based on code by Professor Ross.
#include <iostream>
#include <iomanip>
#include <string.h>
#include <fstream>
#include <stdlib.h>
#include <time.h>
#include <string>

using namespace std;

//Original sub tables held here.
unsigned char f[4][256] = {    { 1,  18, 121,  32, 127, 137, 132, 136, 144, 152, 159, 167,
175, 176, 178,   6, 149, 179, 186, 112,  59,  64,  12,   9,  81,  41,  29,  16,  10,   5,   0,
31, 76, 115, 165, 168,  74, 105,  48, 124, 172,  51, 182, 195,  63, 123,  38, 200, 205, 210,
28,  53, 212, 215, 217, 220, 109,  72,  99, 148, 158, 222, 229, 230,
    57,  90, 104,  55,   8, 131, 169,  87, 180, 189, 213, 214, 194,  85, 133, 187,  73,  58,
66,  54, 141,  86,  15,  60,  82, 142,  47, 111, 157,  19,  36, 163,
    166, 184,  71, 199, 201, 100,  39,  40,  95, 101, 139, 155,  62,  33, 177, 202,  25, 216,
223, 150, 227, 232, 235, 237, 238,  43, 240, 241,   2, 183, 122, 236,
    24, 130, 161, 171,  23,  69, 181,  44, 190, 174, 219,  49,  75, 135, 208, 225, 234, 242,
243, 244,  67,  65, 146, 245, 239,  98,   7, 147, 120, 203, 207, 145,
    80, 221, 160, 107,  61,  22,  92, 143, 193,  35, 185, 196, 204, 224, 231, 246, 248, 249,
250,  52, 251, 192, 197, 103, 119,  37,  77,  21, 129, 151, 170, 191,
    198, 206, 209, 102,  11, 153,  88, 164, 226, 228, 116, 134, 233, 253, 126, 162,  42,   3,
138, 154,  78, 218, 247, 252, 211, 254,  96,  68, 110,  83, 125,  34,
    128,  97, 106, 255, 156,   4,  26,  27,  30, 108,  56,  13,  93, 140,  17,  45, 173,  50,
118, 188,  91, 114,  46,  84,  89,  20,  14,  79,  94,  70, 113, 117 },

    { 112, 103, 224, 128, 140, 156,  60, 219,  25,  44, 213, 180, 187, 171, 169, 197, 228,
221, 236,  84, 164, 117, 225,  71,   1,  85,  27,  87,  99, 249, 118, 250, 214, 106, 239,  57,
92,  20, 138, 148, 181,  21, 185,  46, 226, 159, 240, 251,   2, 166, 252,  82, 253, 254,  65,
4, 105,  39, 177, 231, 255,  12,  64, 170,
    53,  36, 182,  26,  22,  42,   9,  54, 119, 137, 124,  37, 155,  31,  86,  35, 107, 186,
216, 101,  33,  69, 116, 136, 134,  30,  49, 142, 143, 160, 179, 193,
    220, 235, 176, 131,   0,  61,  47,  81, 108, 233,  18,  11,  28,  45,  95,  98, 162, 174,
175, 125,  67, 120, 194, 102, 201, 211,   8,  38,  62, 158, 163, 237,
    3,  79, 150, 205, 238,  58, 241,  68, 178,  40,  14, 114, 123, 129, 144, 195, 203, 243,
100, 199,  76, 157,  50, 167,  24, 208, 212, 196, 217, 121,  41,  59,
    115, 135, 127, 149, 184, 190,  29,  97, 189, 215,  90, 133,  83,  72, 244, 248,  94, 152,
23,  89, 139, 191, 200,  32,  91, 202, 227,  48, 209,  75,  80,   7,
    154, 210, 126, 141, 232,  70, 218, 165, 198,  55, 110, 147,   5,   6,  51, 153, 173, 204,
161, 206,  43, 207, 188,  19, 223, 234,  10,  63,  52,  66,  34,  93,
    130, 242,  88, 183,  13, 146, 230, 229,  17, 132, 168, 245, 246, 145, 111,  74,  16,  56,
104, 247, 109, 113,  73, 192, 222,  77, 172,  15, 122, 151,  78,  96 },

    { 71,  68,  46, 109, 126, 147, 179,  62, 183, 195, 213,  67,  83, 191,  11,  37, 118, 193,
216,  14, 186,  73, 101, 114, 140, 142, 231,   0, 205, 232,  12,  40, 123,  32, 137,  23,  25,
38, 132, 143,  27, 107, 121,  94, 203,  82, 131, 163, 196, 206, 149, 218,  50,  72,  81, 146,
111, 100, 219, 110,  75, 127,  44, 182,
    155,  48,  76, 189,  79, 210,  56, 116, 164, 198,  20, 209, 214, 113, 138,   1, 220, 221,
34, 158, 225, 229,  86,  98, 234,   3,  42, 236, 237, 238, 241, 243,
    16,  57,   9,   8,  28,  33,  13,   7,  80,  96, 103, 159,  17, 228, 150, 177, 184, 212,
239, 141, 244, 144,  90,  78, 171, 106, 152,  55, 176,  85, 122,  10,
    54, 102,  26,  60, 139, 148, 153,  53,   4,  65,  84, 161, 169,  49, 187,  58, 222, 224,
226, 130, 136,  43, 245, 207,  35, 157, 246, 248, 105,  21, 249, 250,
    251, 252, 253, 133,  41,  47,   6, 247, 254, 240, 128, 162,  63,  97, 173,  66,  15, 104,
69, 108, 119,  52, 129, 167, 170, 115,  59, 175, 180, 185, 145,  89,
    135, 181,  22, 165, 112,   5, 124, 160,  87, 192, 200,  92, 201, 215,  30, 217,  51, 174,
223, 230,  70, 233, 227, 235, 178, 197,  99, 204, 242,  19, 255,  93,
    64,  29,  18,  39,  95,  45, 151, 188, 190, 117, 194,  74, 199,  88, 168, 202, 125, 154,
208,  36, 134, 166,  77,  31, 156, 172,   2, 211, 120,  61,  24,  91 },

    { 224, 102, 161,  74, 213, 234, 212,  69, 246,  30,  64, 107,  66,  81,  82, 123,  76,
132, 112, 136, 105, 129, 194, 151, 215,  40, 120,  49,  83, 148, 100, 131, 15,  65,  43, 134,
174, 189, 216, 138, 220, 214, 223, 225, 142,  71, 205,  48, 217, 227,  34,  12, 169,  17, 179,
237, 238, 218,   3,  16,  41,  61, 109,  45,
    198, 239, 119,  92,  25, 230, 243, 244, 247, 248, 249, 250, 150, 253,  58, 236,   8,  94,
121, 167, 251,  52,  44,  97,  21,  36,   5,  42, 114,  32,  95, 166,
    196, 201, 219,  26, 101,  57,  37, 140,  46,   7, 125,  77, 153, 171, 187, 203, 204,  56,
70, 207,  67, 115, 135, 173,  90, 195, 197, 209, 210, 211, 192, 221,
    226, 157, 162,  20,  96, 147, 199,  98, 126,  80, 178, 182, 228, 103, 229, 128, 156,  13,
144,  23,  51, 139, 172, 183,  63, 110, 202, 208, 231,  35, 235, 155,
    241, 252,  27, 254, 255,  68, 133,   9,  62,   1,  99,  87, 108, 130, 137,  54,  84, 149,
0, 104, 113,  93, 127, 145,  79, 154,  14, 164,  22, 143, 170, 177,
    180, 181, 163, 186,  60, 124, 191, 193, 190,  33,  53, 175,  91, 240, 117, 242, 245, 176,
185,  29,  73, 152, 233,  18, 111, 116, 146,  75,  78,  89, 141, 158,
    168, 200, 222,  50,  85,  88, 184,  86,  55, 232, 122,   6,  10,  38,   4,  39, 106, 118,
2,  19,  47, 160,  59,  24,  11,  31,  72, 165, 188, 206,  28, 159 }
};

//Inverse sub tables held her.
unsigned char fi[4][256];
```

```
// fill and print the inverse function table
void fill_fi()
{
    //This code inverses the substitution tables.
    int iTemp = 0;

    for (int count = 0; count < 4; count++)
    {
        for (int index = 0; index < 256; index++)
        {
               iTemp = f[count][index];

               fi[count][iTemp] = index;
        }
    }

    //This code prints out all 4 inverse substitution tables.
    for (int count = 0; count < 4; count++)
    {
        cout << "{ ";
        for (int index = 0; index < 256; index++)
        {
               printf("%d", fi[count][index]);
               cout << ", ";
        }
        cout << "}; " << endl << endl;
    }
}

//Swaps the high and low nibbles of a byte
unsigned char swapbytes(unsigned char cIn)
{
    unsigned char lownibble, highnibble, cOut = 0;

    lownibble = cIn & 0x0F;
    highnibble = cIn & 0xF0;

    cOut = highnibble >> 4;
    cOut = cOut | (lownibble << 4);

    return cOut;
}

//This function decrypts a unsigned char given a certain key.
unsigned char decrypt(unsigned char w, unsigned char key)
{
    //Intermediate values in the process
    unsigned char x0, y0, z0;
    unsigned char x1, y1, z1;
    unsigned char x2, y2, z2;
    unsigned char x3, y3, z3;

    //Get base_4 digit values of key by parsing bits... every 2 bits is a base_4 digit
    //Key = s x 4^3  +  r x 4^2  +  q x 4^1  +  p x 4^0
    unsigned char p, q, r, s;
    p = (key & 0x03);
    q = (key & 0x0C) >> 2;
    r = (key & 0x30) >> 4;
    s = (key & 0xC0) >> 6;

    //Stage 0
    x0 = w ^ key;      //XOR
    y0 = swapbytes(x0); //Transposition
    z0 = fi[p][y0];           //Substitution


    //Stage 1
    x1 = z0 ^ key;
    y1 = swapbytes(x1);
    z1 = fi[q][y1];

    //Stage 2
    x2 = z1 ^ key;
    y2 = swapbytes(x2);
    z2 = fi[r][y2];

    //Stage 3
    x3 = z2 ^ key;
    y3 = swapbytes(x3);
    z3 = fi[s][y3];

    return z3;
}

//This function encrypts a unsigned char given a certain key.
unsigned char encrypt(unsigned char w, unsigned char key)
{
    //Intermediate values in the process
    unsigned char x0, y0, z0;
    unsigned char x1, y1, z1;
    unsigned char x2, y2, z2;
    unsigned char x3, y3, z3;

    //Get base_4 digit values of key by parsing bits... every 2 bits is a base_4 digit
    //Key = s x 4^3  +  r x 4^2  +  q x 4^1  +  p x 4^0
    unsigned char p, q, r, s;
    p = (key & 0x03);
    q = (key & 0x0C) >> 2;
    r = (key & 0x30) >> 4;
    s = (key & 0xC0) >> 6;

    //Stage 0
    x0 = f[s][w];     //Substitution
    y0 = swapbytes(x0); //Transposition
    z0 = y0 ^ key;    //XOR

    //Stage 1
    x1 = f[r][z0];
    y1 = swapbytes(x1);
    z1 = y1 ^ key;

    //Stage 2
    x2 = f[q][z1];
    y2 = swapbytes(x2);
    z2 = y2 ^ key;

    //Stage 3
    x3 = f[p][z2];
    y3 = swapbytes(x3);
    z3 = y3 ^ key;

    return z3;
}

//Main function to exeute.
void main()
{
    //Filling the inverse sub tables.
    fill_fi();

    //This byte will be encrypted.
    char c;

    //This byte holds the key used for encryption.
    unsigned char key = 42;

    //Open file to be encrypted.
    ifstream originalFile("original.txt", ios_base::binary);

    //Open file where the new encrypted file will be written.
    ofstream encryptFile("encrypt.txt", ios_base::binary);

    //Open file where the new decrypted file will be written.
    ofstream decryptFile("decrypt.txt", ios_base::binary);

    //Read, Encrypt, Write, Decrypt.
    while (!originalFile.eof())
    {
        originalFile.read(&c, 1);
        if (!originalFile.eof())
        {
                //Note, the character is encrypted and then written to a file.
                //The same character is then decrypted and written to a different file.
                c = encrypt(c, key);
                encryptFile.write(&c, 1);
                c = decrypt(c, key);
                decryptFile.write(&c, 1);
        }
    }
    originalFile.close();
    encryptFile.close();
    decryptFile.close();
}
```

Output:

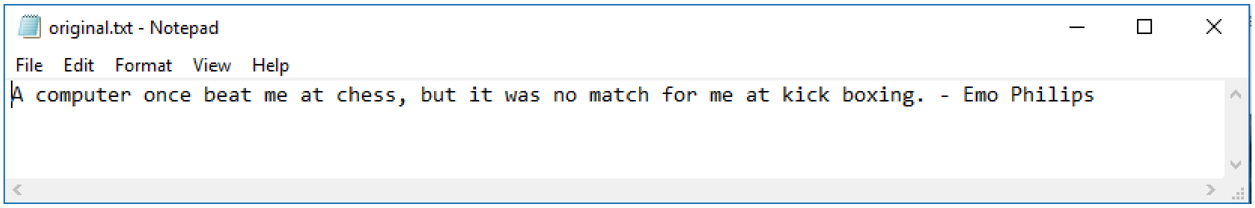
Note: The first output will contain the inverse substitution tables. The outputs after that will be images of the text files that were used/created during the programs runtime.



```
{ 30, 0, 124, 209, 229, 29, 15, 154, 68, 23, 28, 196, 22, 235, 250, 86, 27, 238, 1, 93, 249, 187, 165, 132, 128, 112, 23
0, 231, 50, 26, 232, 31, 3, 109, 223, 169, 94, 185, 46, 102, 103, 25, 208, 121, 135, 239, 246, 90, 38, 139, 241, 41, 179
, 51, 83, 67, 234, 64, 81, 20, 87, 164, 108, 44, 21, 149, 82, 148, 219, 133, 253, 98, 57, 80, 36, 140, 32, 186, 212, 251
, 160, 24, 88, 221, 247, 77, 85, 71, 198, 248, 65, 244, 166, 236, 252, 104, 218, 225, 153, 58, 101, 105, 195, 183, 66, 3
7, 226, 163, 233, 56, 220, 91, 19, 254, 245, 33, 202, 255, 242, 184, 156, 2, 126, 45, 39, 222, 206, 4, 224, 188, 129, 69
, 6, 78, 203, 141, 7, 5, 210, 106, 237, 84, 89, 167, 8, 159, 150, 155, 59, 16, 115, 189, 9, 197, 211, 107, 228, 92, 60,
10, 162, 130, 207, 95, 199, 34, 96, 11, 35, 70, 190, 131, 40, 240, 137, 12, 13, 110, 14, 17, 72, 134, 42, 125, 97, 170,
18, 79, 243, 73, 136, 191, 181, 168, 76, 43, 171, 182, 192, 99, 47, 100, 111, 157, 172, 48, 193, 158, 142, 194, 49, 216,
 52, 74, 75, 53, 113, 54, 213, 138, 55, 161, 61, 114, 173, 143, 200, 116, 201, 62, 63, 174, 117, 204, 144, 118, 127, 119
, 120, 152, 122, 123, 145, 146, 147, 151, 175, 214, 176, 177, 178, 180, 215, 205, 217, 227, };

{ 100, 24, 48, 128, 55, 204, 205, 191, 122, 70, 218, 107, 61, 228, 138, 251, 240, 232, 106, 215, 37, 41, 68, 178, 152, 8
, 67, 26, 108, 166, 89, 77, 183, 84, 222, 79, 65, 75, 123, 57, 137, 158, 69, 212, 9, 109, 43, 102, 187, 90, 150, 206, 22
0, 64, 71, 201, 241, 35, 133, 159, 6, 101, 124, 219, 62, 54, 221, 116, 135, 85, 197, 23, 173, 246, 239, 189, 148, 249, 2
54, 129, 190, 103, 51, 172, 19, 25, 78, 27, 226, 179, 170, 184, 36, 223, 176, 110, 255, 167, 111, 28, 146, 83, 119, 1, 2
42, 56, 33, 80, 104, 244, 202, 238, 0, 245, 139, 160, 86, 21, 30, 72, 117, 157, 252, 140, 74, 115, 194, 162, 3, 141, 224
, 99, 233, 171, 88, 161, 87, 73, 38, 180, 4, 195, 91, 92, 142, 237, 229, 203, 39, 163, 130, 253, 177, 207, 192, 76, 5, 1
49, 125, 45, 93, 210, 112, 126, 20, 199, 49, 151, 234, 14, 63, 13, 250, 208, 113, 114, 98, 58, 136, 94, 11, 40, 66, 227,
 164, 42, 81, 12, 214, 168, 165, 181, 247, 95, 118, 143, 155, 15, 200, 147, 182, 120, 185, 144, 209, 131, 211, 213, 153,
 188, 193, 121, 154, 10, 32, 169, 82, 156, 198, 7, 96, 17, 248, 216, 2, 22, 44, 186, 16, 231, 230, 59, 196, 105, 217, 97
, 18, 127, 132, 34, 46, 134, 225, 145, 174, 235, 236, 243, 175, 29, 31, 47, 50, 52, 53, 60, };

{ 27, 79, 250, 89, 136, 197, 166, 103, 99, 98, 127, 14, 30, 102, 19, 176, 96, 108, 226, 221, 74, 157, 194, 35, 254, 36,
130, 40, 100, 225, 206, 247, 33, 101, 82, 152, 243, 15, 37, 227, 31, 164, 90, 149, 62, 229, 2, 165, 65, 141, 52, 208, 18
1, 135, 128, 123, 70, 97, 143, 186, 131, 253, 7, 172, 224, 137, 175, 11, 1, 178, 212, 0, 53, 21, 235, 60, 66, 246, 119,
68, 104, 54, 45, 12, 138, 125, 86, 200, 237, 191, 118, 255, 203, 223, 43, 228, 105, 173, 87, 218, 57, 22, 129, 106, 177,
 156, 121, 41, 179, 3, 59, 56, 196, 77, 23, 185, 71, 233, 16, 180, 252, 42, 126, 32, 198, 240, 4, 61, 170, 182, 147, 46,
 38, 163, 244, 192, 148, 34, 78, 132, 24, 115, 25, 39, 117, 190, 55, 5, 133, 50, 110, 230, 122, 134, 241, 64, 248, 153,
83, 107, 199, 139, 171, 47, 72, 195, 245, 183, 238, 140, 184, 120, 249, 174, 209, 187, 124, 111, 216, 6, 188, 193, 63, 8
, 112, 189, 20, 142, 231, 67, 232, 13, 201, 17, 234, 9, 48, 217, 73, 236, 202, 204, 239, 44, 219, 28, 49, 151, 242, 75,
69, 251, 113, 10, 76, 205, 18, 207, 51, 58, 80, 81, 144, 210, 145, 84, 146, 214, 109, 85, 211, 26, 29, 213, 88, 215, 91,
 92, 93, 114, 169, 94, 220, 95, 116, 150, 154, 167, 155, 158, 159, 160, 161, 162, 168, 222, };

{ 178, 169, 242, 58, 238, 90, 235, 105, 80, 167, 236, 248, 51, 145, 186, 32, 59, 53, 215, 243, 131, 88, 188, 147, 247, 6
8, 99, 162, 254, 211, 9, 249, 93, 201, 50, 157, 89, 102, 237, 239, 25, 60, 91, 34, 86, 63, 104, 244, 47, 27, 227, 148, 8
5, 202, 175, 232, 113, 101, 78, 246, 196, 61, 168, 152, 10, 33, 12, 116, 165, 7, 114, 45, 250, 212, 3, 219, 16, 107, 220
, 184, 137, 13, 14, 28, 176, 228, 231, 171, 229, 221, 120, 204, 67, 181, 81, 94, 132, 87, 135, 170, 30, 100, 1, 141, 179
, 20, 240, 11, 172, 62, 153, 216, 18, 180, 92, 117, 217, 206, 241, 66, 26, 82, 234, 15, 197, 106, 136, 182, 143, 21, 173
, 31, 17, 166, 35, 118, 19, 174, 39, 149, 103, 222, 44, 189, 146, 183, 218, 133, 29, 177, 76, 23, 213, 108, 185, 159, 14
4, 129, 223, 255, 245, 2, 130, 194, 187, 251, 95, 83, 224, 52, 190, 109, 150, 119, 36, 203, 209, 191, 138, 54, 192, 193,
 139, 151, 230, 210, 195, 110, 252, 37, 200, 198, 126, 199, 22, 121, 96, 122, 64, 134, 225, 97, 154, 111, 112, 46, 253,
115, 155, 123, 124, 125, 6, 4, 41, 24, 38, 48, 57, 98, 40, 127, 226, 42, 0, 43, 128, 49, 140, 142, 69, 156, 233, 214, 5,
 158, 79, 55, 56, 65, 205, 160, 207, 70, 71, 208, 8, 72, 73, 74, 75, 84, 161, 77, 163, 164, };

Press any key to continue . . .
```

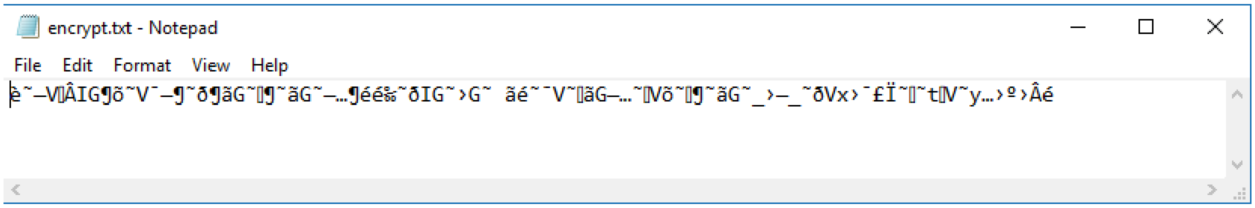
File Test #1:

Original:



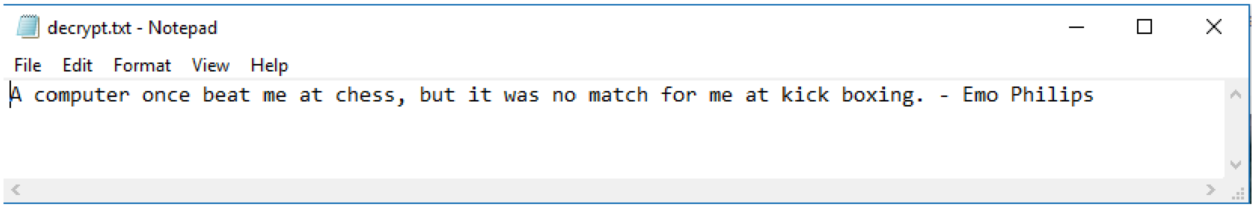
```
A computer once beat me at chess, but it was no match for me at kick boxing. - Emo Philips
```

Encrypted:



```
è˜–V▯ÂIG¶õ˜V¯–¶˜ð¶ãG˜▯¶˜ãG˜–…¶éé‰˜ðIG˜›G˜ ãé˜¯V˜▯ãG–…˜▯Võ˜▯¶˜ãG˜_›–_˜ðVx›¯£Ï˜▯˜t▯V˜y…›º›Âé
```

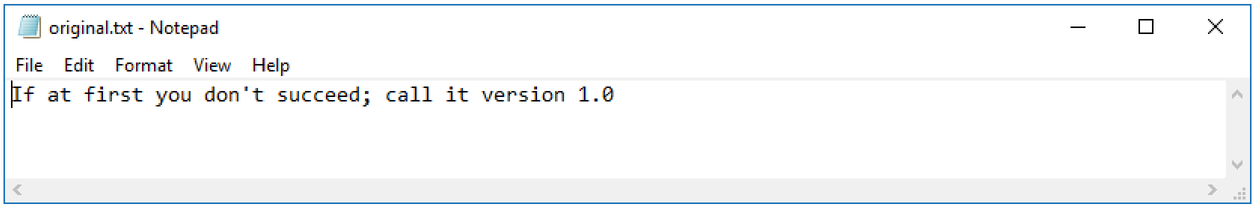
Decrypted:



```
A computer once beat me at chess, but it was no match for me at kick boxing. - Emo Philips
```

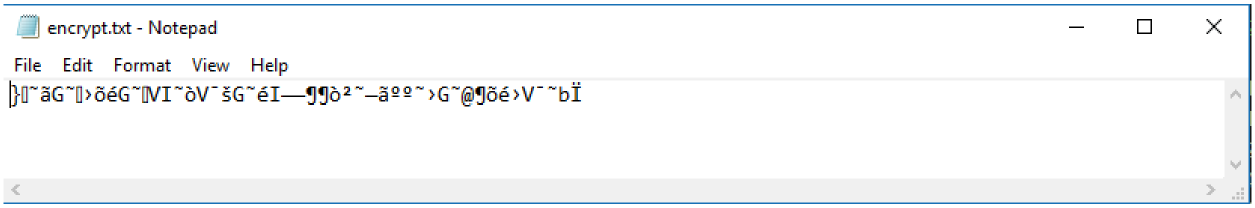
File Test #2:

Original:



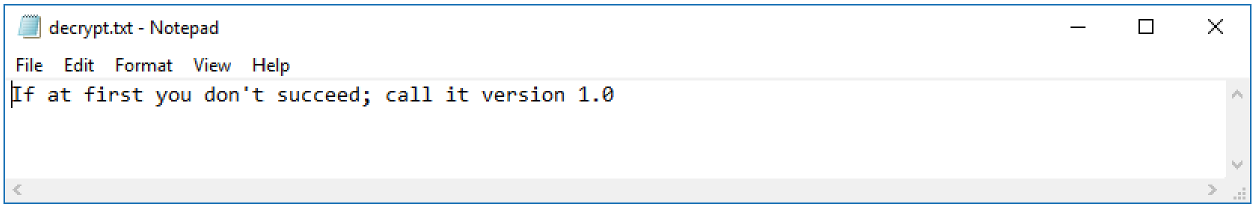
```
If at first you don't succeed; call it version 1.0
```

Encrypted:



```
}▯˜ãG˜▯›õéG˜▯VI˜òV¯šG˜éI—¶¶ò²˜–ãºº˜›G˜@¶õé›V¯˜bÏ
```

Decrypted:



```
If at first you don't succeed; call it version 1.0
```

Part 1 - Decrypt a Bitmap

Description:

For this section of the assignment our goal was to take an encrypted bitmap image provided by our professor and find the key to decrypt. To do this, I modified the code that was in main to iterate through all 255 possible keys.

Note: Under source code I just listed the changes made to main to allow for the code to iterate through the keys.

Source Code:

```
//Main function to exeute.
void main()
{
    //Filling the inverse sub tables.
    fill_fi();

    //This byte will be encrypted.
    char c;

    //This byte holds the key used for encryption.
    unsigned char key = 42;

    for (int count = 0; count <= 255; count++)
    {
        string name = to_string(count) + ".bmp";

        ifstream file("ePic.bmp", ios_base::binary);
        ofstream outFile(name, ios_base::binary);

        key = count;

        while (!file.eof())
        {
                file.read(&c, 1);
                if (!file.eof())
                {
                       c = decrypt(c, key);
                       outFile.write(&c, 1);
                }
        }
        file.close();
        outFile.close();
    }
}
```

Output:

The output for this process was a grand total of 255 bitmap files. However, number 177 was different from the rest. This file was actually a displayable image of a pizza. Thus, the key was 177 (base 10) or B1 (hex).

(177.bmp)

Conclusion

This assignment was interesting. After examining the code and thinking about how the encryption process worked I was able to quickly write up a function that decrypted what the code encrypted. I found the process to be intriguing and enjoyed messing around with different text and files after my code was working. Looking forward to what's next.