COMP 2401 -- Assignment #1

Due: Thursday, October 9, 2014 at 9:00 PM

Goal

You will write a program in C, in the Ubuntu Linux environment, to provide the end user with the ability to encrypt and decrypt messages, using a secret key. Your program will prompt the user to indicate whether they wish to encrypt a readable ("plaintext") message or decrypt an already encrypted sequence of numbers ("ciphertext") back to its readable form. You will implement the encryption and decryption algorithm described in the instructions below. In order to prevent your enemies from easily cracking your encrypted messages, the algorithm will use an incrementing counter to encrypt each character in a slightly different way. Your program will output the result (either the ciphertext or the plaintext) to the screen. You will begin with the skeleton code found here.

Learning Objectives

- familiarize yourself with the Linux programming environment
- write a small program in C that is modular, correctly designed and documented
- manipulate values at the bit level

Instructions

1. **get/set/clear bit functions**

   Implement the **`getBit`** , **`setBit`** and **`clearBit`** functions, exactly as we saw in class.

2. **Encode function**

   Implement the **encode** function, which works for both encryption and decryption, i.e. the same algorithm is used for translation in both directions. This function takes one *source byte* (either plaintext or ciphertext), a key, and an integer counter, and it returns the corresponding encrypted or decrypted *resulting byte*.

   The **encode** function must have the following prototype:

   ```
   unsigned char encode(unsigned char source, unsigned char key, int counter)
   ```

   The function uses the following algorithm:

   - begin with your resulting byte assigned the value of the **`source`** byte
   - if the **`counter`** modulo 3 is zero, examine *every **other** bit* of both the **`source`** and **`key`** bytes, starting at bit 0
     - o if the **`source`** bit **`xor`** the **`key`** bit is zero, set the same bit in the resulting byte to zero
     - o if the **`source`** bit **`xor`** the **`key`** bit is one, set the same bit in the resulting byte to one
   - if the **`counter`** modulo 3 is one, examine *every **other** bit* of both the **`source`** and **`key`** bytes, starting at bit 1
     - o if the **`source`** bit **`xor`** the **`key`** bit is zero, set the same bit in the resulting byte to zero
     - o if the **`source`** bit **`xor`** the **`key`** bit is one, set the same bit in the resulting byte to one
   - if the **`counter`** modulo 3 is two, examine *every bit* of both the **`source`** and **`key`** bytes, starting at bit 0
     - o if the **`source`** bit **`xor`** the **`key`** bit is zero, set the same bit in the resulting byte to zero
     - o if the **`source`** bit **`xor`** the **`key`** bit is one, set the same bit in the resulting byte to one

   Note:

   - you **must** use the **`getBit`**, **`setBit`** and **`clearBit`** functions everywhere applicable

3. Encryption and decryption

Fill in the missing code in the **`main`** function of the provided skeleton code:

- if the user chooses to encrypt a message, the program:
  - prompts the user to enter a plaintext message
  - prompts the user to enter a numeric key (remember the valid range for **`unsigned char`**s!)
  - loops over each byte of the plaintext and encodes it with the key
    - the counter begins at zero and increments by one with each iteration
    - the resulting byte is the encrypted plaintext byte
  - prints the encrypted message as a sequence of numbers to the screen
- if the user chooses to decrypt a message, the program:
  - prompts the user to enter an encrypted message (ciphertext) as a sequence of numbers terminated by -1
  - prompts the user to enter a numeric key (remember the valid range for **`unsigned char`**s!)
  - loops over each byte of the ciphertext and encodes it with the key
    - the counter begins at zero and increments by one with each iteration
    - the resulting byte is the decrypted ciphertext byte
  - prints the decrypted message to the screen

Notes:

- remember! you must use the **same** key value to decrypt a message as was used to encrypt it
- your program must perform all basic error checking (e.g. range checking)
- you can find some helpful sample code for safe user I/O here: testUtil.c

Testing: You will know that your program works correctly if it can decrypt the following sequence using key 178

```
88 205 197 48 207 211 126 219 146 96 208 221 119 208 211 125 207 215 98 209 146 116
205 215 99 130 219 100 130 198 113 201 215 48 214 221 48 193 218 113 204 213 117 130
211 48 206 219 119 202 198 48 192 199 124 192 141 48 236 221 126 199 146 242 34 33 48
235 198 242 34 43 99 130 211 48 202 211 98 198 197 113 208 215 48 210 192 127 192 222
117 207 -1
```

Constraints

- you must use the skeleton program provided and use the function prototypes exactly as stated
- do **not** use any global variables
- you must use **all** the functions that were provided or that you implemented
- you must reuse functions everywhere possible
- your program must be thoroughly commented
- programs that do not compile, do not execute, or violate any constraint are subject to severe deductions
- late assignments are never accepted

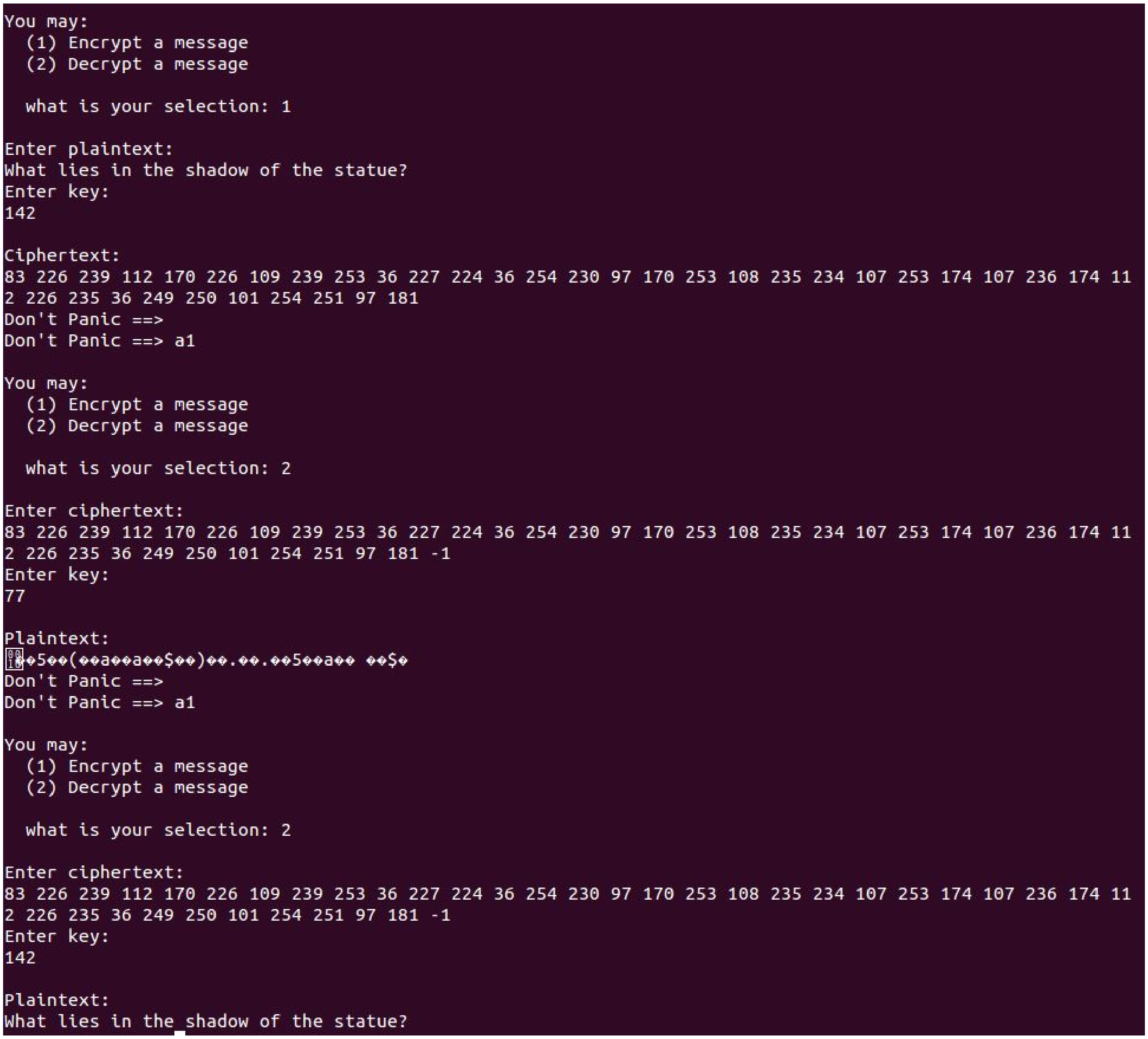
Submission

You will submit in *cuLearn*, before the due date and time, one **`tar`** file that includes all the following:

- all source code
- a readme file, which must include:
  - a preamble (program author, purpose, list of source/header/data files)
  - exact compilation command
  - launching and operating instructions

Sample Output

You can view a demo of the program here. Below is the output of a sample execution:



```
You may:
  (1) Encrypt a message
  (2) Decrypt a message

  what is your selection: 1

Enter plaintext:
What lies in the shadow of the statue?
Enter key:
142

Ciphertext:
83 226 239 112 170 226 109 239 253 36 227 224 36 254 230 97 170 253 108 235 234 107 253 174 107 236 174 11
2 226 235 36 249 250 101 254 251 97 181
Don't Panic ==>
Don't Panic ==> a1

You may:
  (1) Encrypt a message
  (2) Decrypt a message

  what is your selection: 2

Enter ciphertext:
83 226 239 112 170 226 109 239 253 36 227 224 36 254 230 97 170 253 108 235 234 107 253 174 107 236 174 11
2 226 235 36 249 250 101 254 251 97 181 -1
Enter key:
77

Plaintext:
��5���(��a���a���$���)��.��.���5���a�� ���$�
Don't Panic ==>
Don't Panic ==> a1

You may:
  (1) Encrypt a message
  (2) Decrypt a message

  what is your selection: 2

Enter ciphertext:
83 226 239 112 170 226 109 239 253 36 227 224 36 254 230 97 170 253 108 235 234 107 253 174 107 236 174 11
2 226 235 36 249 250 101 254 251 97 181 -1
Enter key:
142

Plaintext:
What lies in the shadow of the statue?
```

Grading

- **Marking breakdown:** The grading scheme is posted in cuLearn.
- **Deductions:**
  - Up to 50 marks for any of the following:
    - the code does not compile using gcc in the VM provided for the course
    - unauthorized changes have been made to the skeleton code provided for you
    - code cannot be tested because it doesn't run
  - Up to 20 marks for any of the following:
    - your program is not broken down into multiple reusable, modular functions
    - your program uses global variables (unless otherwise explicitly permitted)
    - the readme file is missing or incomplete
  - Up to 10 marks for missing comments or other bad style (indentation, identifier names, etc.)
- **Bonus marks:**
  - Up to 5 extra marks are available for fun and creative additional features