An Exclusive

1. The recipient should use XOR to decrypt the ciphertext. When the ciphertext is XOR'd with the same one-time pad used to encrypt the message, the recipient will get the original message.

```
  11100010    00011100
^ 10101010    01010101
----------------------------------
  01001000    01001001
```

2. The original text can be obtained by XOR'ing the ciphertext and the one-time pad as follows:

	01100101	10100010	01101111
^	00111100	11100111	00111100
	01011001	01000101	01010011
	89	69	83
ASCII	Y	E	S

   Therefore, the three ASCII characters are Y, E, and S.

3. We can encrypt the longer message by using a key, which is generated by repeatedly concatenating the original key of 16 bits. For example, if the message is 64 bits long, then our new key will be generated by concatenating the original key of 16 bits 4 times. If the message is not a multiple of 16, for example if it is 72 bits long, then our new key will be generated by concatenating the original key 4 times and its left most 8 bits to produce a new key of 72 bits.

4. A longer key is more secure, because a ciphertext generated using a shorter XOR key can be decrypted using frequency analysis of characters. Since the frequency distribution of letters cannot be effectively masked using a shorter XOR key, using the typical distribution of English letters, one can break such ciphertexts. In addition, when a short XOR key is repeatedly used to encrypt a longer message, if one part of the message can be guessed, then the remaining part of the message can be easily broken.

5. The given function switches the values at the addresses pointed to by a and b. In other words, if the initial values of *a and *b are 1 and 2 respectively, the new values of *a and *b after the function is called will be 2 and 1 respectively. A better name for this function would be "swap".

6. If two out of the three drives fail, then we cannot recover those two drives. In this case, we only have one good drive.

7. We only need one additional hard drive. We need to XOR the data from the first three drives and store the results onto the fourth drive. For example, if the data in the first three drives are namely A, B, and C, then we will store D = A^B^C onto the fourth drive (backup drive). Supposing one of the three drives fails, we can XOR the remaining two drives with our backup drive to recover the lost drive. This is because, since D = A^B^C, we know that A = D^B^C, B = D^A^C, and C = D^A^B.

Debrief

1. Which resources, if any, did you find helpful in answering this problem's questions? Wikipedia (XOR cipher article)

2. 80 minutes

Assembly Line

1. The function of each line in the program is listed below. The numbers correspond to the line numbers in the program:
    1. This statement sets the value of register r1 to string “x: “.
    2. Displays the value of the contents of register r1. This will display “x: “ on the screen.
    3. This statement prompts the user for an input value and when the user inputs a value, stores that value in register r2.
    4. This statement sets the value of register r1 to string “y: “.
    5. Displays the value currently stored in register r1. In other words, it will display “y:” on the screen.
    6. This statement prompts the user for an input value. When the user inputs a value, it then stores that value in register r3.
    7. This statement evaluates whether the contents of the registers r2 and r3 are equal. If they are equal, the program jumps to line 11.
    8. Note that we reach this line only if the JUMPEQ evaluation in Line 7 fails. In this case, the content of the register r1 is set to the string, “x is not equal to y”.
    9. This statement displays the current content stored in register r1. In other words, it displays “x is not equal to y”.
    10. The program exits execution.
    11. The program will only reach this line if the JUMPEQ statement on Line 7 evaluates true. In other words, the program will reach this point only if x is equal to y. This statement sets the value of register, r1, to the string “x is equal to y”.
    12. This statement prints the contents of the register r1. Therefore, the string “x is equal to y” is displayed.
    13. The program exits execution.

2. The following program, when the two numbers are not equal, instead of just printing out “x is not equal to y”, it prints either “x is less than y” or “x is greater than y”, depending on which number is greater. It still prints “x is equal to y” I the two numbers are equal.
    1. SET r1 “x: “
    2. PRINT r1
    3. INPUT r2
    4. SET r1 “y: “
    5. PRINT r1
    6. INPUT r3
    7. JUMPEQ r2 r3 15
    8. JUMPLT r2 r3 12
    9. SET r1 “x is greater than y”
    10. PRINT r1
    11. EXIT
    12. SET r1 “x is less than y”
    13. PRINT r1
    14. EXIT

15. SET r1 “x is equal to y”
16. PRINT r1
17. EXIT

3. The following program “coughs” some (n) number of times input by the user.
    1. SET r1 “n: “
    2. PRINT r1
    3. INPUT r2
    4. SET r3 0
    5. SET r1 1
    6. SET r4 “cough”
    7. PRINT r4
    8. ADD r3 r3 r1
    9. JUMPLT r3 r2 6
    10. EXIT

4. The name of the instruction for calling a function is **callq**. This is because in the original C program, we are calling the function **`get_int()`** twice, and in the assembly code, we see that the instruction callq is followed by **`get_int()`** twice. Similarly, in the original C program, there are two calls to the **`printf()`** function, and we note that the assembly code has two occurrences of **callq** followed by **`printf().`** This indicates that **callq** is the x86-64 assembly instruction for calling a function.

5. The name of the x86-64 instruction that determines what to print is **jge**, which means “jump if greater than or equal to”. The preceding instruction to **jge** in the assembly code is **cmpl**, which compares the contents (values of **`x`** and **`y`**) of the two registers **`rbp`** and **`eax`** and sets one of the bits in condition register field depending on the values of **`x`** and **`y`**. Based on the conditional bit, **jge** performs a comparison jump if the destination operand is greater than or equal to the source operand. From the assembly code, we note that if the call to **jge** is successful, it jumps to line number 33 and prints the string **`L.str.3`** (which is, “**`x is not equal to y`**”). (If the conditional jump fails, we print the string **`L.str.2`** (which is, “**`x is less than y`**”).

Debrief

1. Which resources, if any, did you find helpful in answering this problem’s questions?

    X64 Cheat Sheet from the CS department of Brown University

2. About how long, in minutes, did you spend on this problem’s questions?

    60 minutes

Bit by Bitcoin

1. TODO
2. TODO
3. TODO
4.

```
TODO
```

Debrief

1. Which resources, if any, did you find helpful in answering this problem's questions?

TODO

2. About how long, in minutes, did you spend on this problem's questions?

TODO

Count on It

1. We can represent $2^7 = 128$ total characters in ASCII, if each character is represented using 7 bits.

2. The first byte read will tell us how many bytes the character corresponding to that byte occupies. The first byte for a character represented in UTF-8 encoding will have the following format depending on the number of bytes used for that character:

1^{st} Byte	Number of bytes used for the character
`0xxxxxxx`	1
`110xxxxx`	2
`1110xxxx`	3
`11110xxx`	4

   Therefore, depending on the format of the first byte read, we need to further read either 0, 1, 2 or 3 more bytes for a single character. This process repeats until we finish reading the entire file.

3. If the first two bits of a byte are 10, then it is a continuation of an existing character. Therefore, if we XOR the byte with 10000000 (256) and obtain a value less than or equal to 64 (i.e: **`00xxxxxx`**), then the byte is a continuation of an existing character. In other words, a continuation byte is of the form **`10xxxxxx`**.

4. The following program counts the number of characters in a file that is encoded as UTF-8.

   From my answer to Part (2) above, we note that the first byte of a character that is encoded in UTF-8 will be either less than 128 or greater than 128+64 = 192. Therefore, the only check we need to do is to verify that the value of current byte is either less than 128 or greater than or equal to 192.

   Please see the modified program on the next page.

```c
#include <stdbool.h>
#include <stdio.h>

typedef unsigned char BYTE;

int main(int argc, char *argv[])
{
        if (argc != 2)
        {
                printf("Usage: ./count INPUT\n");
                return 1;
        }

        FILE *file = fopen(argv[1], "r");
        if (!file)
        {
                printf("Could not open file.\n");
                return 1;
        }

        int count = 0;
        while (true)
        {
                BYTE b;
                fread(&b, 1, 1, file);
                if (feof(file))
                {
                        break;
                }

                // This may be a UTF-8 encoded file. Therefore, count
                // only the first byte of a character
                //
                If (b < 128 || b >= 192)
                {
                        count++;
                }
        }

        printf("Number of characters: %i\n", count);
}
```

Be sure to submit Question 4 via `submit50` *as well!*

Debrief

1. Which resources, if any, did you find helpful in answering this problem's questions?

    a. Wikipedia article on UTF-8
    b. www.fileformat.info

2. About how long, in minutes, did you spend on this problem's questions?

    60 minutes

File Browser

1.

```
int fgetpos(FILE *stream)
{
    // TODO
}
```

2.

```
bool fsetpos(FILE *stream, int offset)
{
    // TODO
}
```

3.

```
typedef unsigned char BYTE;

bool fwrite(BYTE *buffer, int size, FILE *stream)
{
    // TODO
}
```

4.

```
bool feof(FILE *stream)
{
    // TODO
}
```

Debrief

1. Which resources, if any, did you find helpful in answering this problem’s questions?

TODO

2. About how long, in minutes, did you spend on this problem’s questions?

TODO

Hey, Earl

1. TODO
2. TODO
3.

```
TODO
```

4.

```
TODO
```

5. TODO
6. TODO

Debrief

1. Which resources, if any, did you find helpful in answering this problem's questions?

TODO

2. About how long, in minutes, did you spend on this problem's questions?

TODO

Key Questions

1. Assuming the records for both **`Max`** and **`Ileana`** already exist in table users and their usernames are **`“max”`** and **`“ileana”`** respectively, we need to execute the following SQL statement to insert a row into the followers table:

```
INSERT INTO followers(follower_id, followee_id)
     SELECT VIEW1.id1, VIEW2.id2
          FROM (SELECT id AS id1
                 FROM users
                 WHERE username = "max") VIEW1,
                    (SELECT id AS id2
                 FROM users
                 WHERE username = "ileana") VIEW2;
```

   Please note that in the above query, we could have also used the name column (instead of the username column) in the WHERE clause of the subqueries (example: **`name = “Max”`** instead of **`username = “max”`**).

2. The following query selects the usernames of every user that follows Reese and whom Reese also follows back:

```
SELECT DISTINCT username FROM users, followers,
     (SELECT followers.follower_id AS stalker_id
          FROM users, followers
          WHERE users.username = “reese” AND
                users.id = followers.followee_id) VIEW1
WHERE users.id = followee_id AND
      Followers.followee_id = VIEW1.stalker_id;
```

3. The following query returns the usernames of users who are “two degrees of separation” away from Reese.

```
SELECT DISTINCT users.username FROM users, followers,
     (SELECT DISTINCT followers.followee_id AS friends_id
          FROM users, followers
          WHERE users.username = "reese" AND
                users.id = followers.follower_id) VIEW1
     WHERE VIEW1.friends_id = followers.follower_id AND
          followers.followee_id = users.id AND
          users.username != "reese";
```

4. The following SQL statement creates a **`friendships`** table that allows for the representation of friendships and friend requests. Please note that the column "**`friend`**" is of type INTEGER and can take the values of either 0 or 1, meaning "not a friend" or "friend". (Sqlite does not have a boolean type, which would have been a more appropriate choice for this column.)

```
CREATE TABLE friendships
(
     user_id INTEGER,
     friend_requester_id INTEGER,
     friend INTEGER,
     FOREIGN KEY (user_id) REFERENCES users(id),
     FOREIGN KEY (friend_requester_id) REFERENCES users(id)
);
```

5. The following SQL statement inserts a row into the friendships table when Max sends a friend request to Ileana. Please note that this row is inserted with an initial value of 0 (zero) for the friend column. The initial value of the friend column is 0 indicates that the user specified by **`user_id`** has received a friend request from the user specified by **`friend_requester_id`**.

```
INSERT INTO friendships(user_id,
                        friend_requester_id,
                        friend)
     SELECT VIEW1.id1,
            VIEW2.id2,
            0
     FROM
          (SELECT id AS id1 FROM users
               WHERE username = "max") VIEW1,
          (SELECT id AS id2 FROM users
               WHERE username = "ileana") VIEW2;
```

6. The following SQL statement updates the value of the **`friend`** column in the friendships table for Ileana when she accepts Max's friend request.

```
UPDATE friendships SET friend = 1
     WHERE user_id IN
          (SELECT id from users WHERE username = "ileana")
          AND
          friend_requester_id IN
          (SELECT id from users WHERE username = "max");
```

7. The following three SQL statements deletes all traces of Reese, his friendships, and his followees from the database. Please note that is it important to delete from the **users** table at the end, as the first two queries are dependent on being able to find **Reese** in the **users** table. Please also note that the following queries could also use the **name** column in the **users** table, instead of the **username** column. In this case, the queries will need to use **"Reese"**, instead of **"reese".**

```
DELETE FROM followers
     WHERE follower_id IN
          (SELECT id FROM users
               WHERE username = "reese") OR
          followee_id IN
          (SELECT id FROM users
               WHERE username = "reese");

DELETE FROM friendships
     WHERE user_id IN
          (SELECT id FROM users
               WHERE username = "reese") OR
          friend_requester_id IN
          (SELECT id FROM users
               WHERE username = "reese");

DELETE FROM users WHERE username = "reese";
```

Debrief

1. Which resources, if any, did you find helpful in answering this problem's questions?

   SQLite documentation from www.sqlite.org, and
   Example queries from www.sqlitetutorial.net.
   Sqlite3 database.

2. About how long, in minutes, did you spend on this problem's questions?

   180 minutes.

Less is More

1. The given sequence can be encoded using run-length encoding as: **T3A4C2G1A3**.

2. When the original DNA sequence has each different nucleotide appearing only once. In other words, there are no consecutive occurrences of the same nucleotide in a DNA sequence. For example, the following DNA sequence takes up 6 characters:

   ATGCTA

   However, its run-length encoding takes up 12 characters:

   A1T1G1C1T1A1

3. Flag of Germany could be compressed more. The reason is that the width of the three colors in the flag of Germany is much larger than the width of the three colors in the flag of Romania. Therefore, the number of pixel rows for each color in the flag of Germany is much smaller than that of Romania. Since our compression works by storing just one pixel data followed by the count of the pixels, the flag of Germany will need a smaller number of storage than the flag of Romania.

4. Using the proposed encoding method, **CCCAGTTA** can be encoded as,

   11101011110

5. Using the proposed encoding method, **TGCATCCA** can be encoded as,

   11101011110

   We note that, using the proposed encoding method, we are obtaining identical encoding for different DNA sequences. For example, the nucleotide sequences **CCCAGTTA** and **TGCATCCA** above in Part 4 and 5 result in the same encoding: **11101011110**

   We can improve the above encoding technique by using two bits for each nucleotide. In this new technique, we will use the following encoding:

   - A is represented as 00
   - C is represented as 01
   - G is represented as 10
   - T is represented as 11

   During decoding, the compressed sequence will be decoded from the left end two bits at a time using the above mapping. Therefore, the above encoding will result in unique encoding for distinct DNA sequences. Since each nucleotide takes up only 2 bits as opposed to 8 bits using ASCII, our new approach will use significantly fewer bits.

Debrief

1. None
2. 30 minutes

Random Questions

1. The average running time of **`sorted()`** is O(n log n). Since the comparison of two lists on the average takes O(n) time and the **`issorted()`** function compares two lists using the == operator, the overall average running time of **`issorted()`** is O(n^2 log n).

2. The following implementation of **`issorted()`** has a running time of O(n), as it is comparing each element in the list with the previous element. On the average, it will examine n/2 elements, and in the worst-case, it will examine the entire list.

```
def issorted(numbers):
      size_of_list = len(numbers)

      for i in range(1, size_of_list):
            if numbers[i-1] > numbers[i]:
                  return False
      return True
```

3. The running time of **`issorted()`** cannot ever be O(1), which means a constant number of operations. The reason is that on the average, **`issorted()`** will need to do n/2 comparisons of adjacent elements, and in the worst-case, it will have to compare the adjacent elements in the entire list.

4. In the worst case, the running time of sort will be in the order of O(n!). The reason is that there is a total of n! different ordering for the input list, and in the worst case we will see the ordered list during the very last shuffle. (In that case, the precise worst-case running time will be O(n x n!), but with random shuffling it can take longer).

5. In the best-case scenario, **`itertools.permutations()`** returns the list of numbers in sorted order immediately. Therefore, the lower bound on the running time is O(n).

6. In the worst-case scenario, **`itertools.permutations()`** returns the list of numbers in sorted order only on its last iteration. Therefore, the upper bound on the running time is O(n x n!).

Debrief

1. Which resources, if any, did you find helpful in answering this problem's questions?

   Bogosort on Wikipedia

2. About how long, in minutes, did you spend on this problem's questions?

   30 minutes

Teetering on the Edge

1. TODO
2. TODO
3. TODO
4. TODO
5.

```
TODO
```

Be sure to submit Question 5 via `submit50` *as well!*

Debrief

1. Which resources, if any, did you find helpful in answering this problem's questions?

TODO

2. About how long, in minutes, did you spend on this problem's questions?

TODO