# Cryptogram Part 2

**This assignment is extremely important – (nearly) every assignment after this one uses this one!**

**If you have bugs or missing features in this, you will need to fix them before you can continue on to new assignments. This is very typical in software development outside of school.**

**You must submit .c files. Any other file type will be ignored. Especially ".o" files.**

**You must not zip or otherwise compress your assignment. Brightspace will allow you to submit multiple files.**

***<u>You must submit buildable .c files for credit.</u>***

## Introduction

In this assignment, we will encrypt the puzzle, accept pairs for decryption and show the state of the game correctly.

The data structures we will need for this project are important to get right. We need to have an encrypted version of the string; that string will have to be dynamically allocated (malloc) and freed. We will also need to keep track of the player's progress on decrypting the puzzle. There are many ways to do this, but the most "C-like" takes a little thought.

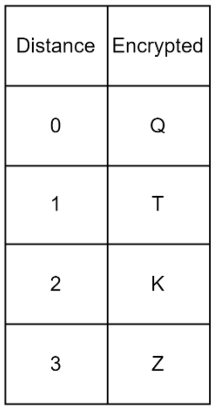
A sentence in English can have upper case letters, lower case letters and "other" (spaces, numbers, etc.). We are only going to encrypt letters. Further, we will convert everything to upper case (this is done even in pen and paper cryptograms). That means we need 26 pairs: A → Q, B → T, etc. Let's call this an **encryption key** – a collection of 26 pairs or mappings.

| Original | Encrypted |
|---|---|
| A | Q |
| B | T |
| C | K |
| D | Z |

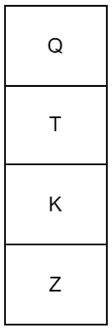
**Original Message** Add Dad

**Encrypted** QZZ ZQZ

| Original | Encrypted |
|---|---|
| 0 | Q |
| 1 | T |
| 2 | K |
| 3 | Z |



| Distance | Encrypted |
|---|---|
| 0 | Q |
| 1 | T |
| 2 | K |
| 3 | Z |



But since the first letter in the pair is incrementing (A,B,C … ) we can "cheat" – we can use the distance from the beginning of the alphabet instead of a letter. That gives us 0 → Q, 1 → T, etc. By doing this, we don't need to STORE the number, since they are predictable – we can use an array of 26 characters for an encryption key. One might ask, though, given a letter, how can I get that distance?

'A'-'A' == 0. 'B' – 'A' == 1. 'C' – 'A' == 2. Any capital letter minus capital A == the distance from the beginning of the alphabet. This is true because we use ASCII in C. In ASCII, A=65, B=66, C=67, etc. This process works both ways:

```
char e = 'A' + 4; // e == 'E'
int distance = e – 'A'; // distance = 4
```

To encrypt the string, we will need to generate an encryption key. One way to do that is to generate the letters in order and then shuffle them. Imagine writing the letters of the alphabet on index cards, so the first is A, the second is B, etc. Then shuffle them. Now, the first is (for example) Q, so A→Q.

To do this in our program, we will use the Fisher-Yates shuffle algorithm:
https://en.wikipedia.org/wiki/Fisher%E2%80%93Yates_shuffle

A quick note on random – computers (and humans) struggle with generating random numbers. Computers do this by running complex math on a starting number. The complex math is designed to make the random numbers evenly distributed but also without a pattern that is easy to detect.

The starting number is called the seed. In C, we will set the seed once near the beginning of our program:

```
srandom(time(NULL)); // set the seed to the number of seconds since
1/1/1970.
```

Then we can use random() and get a random number from 0 to +2 billion. We can narrow that down with modulo.

## Detail

Let’s start with Fisher-Yates. Write a function (shuffle()) that takes the encryption key (an array of 26 char) as a parameter. This function will implement Fisher-Yates to change the order of the letters in the key.

We will need to make two arrays of 26 characters. One for the “real” encryption key and one to hold what the player has entered so far. Since these are fixed in size and small, we don’t need to use malloc() for them – we can make regular old arrays.

Now let’s take a look at initialization(). We will get the puzzle (like before), but now we will generate the encrypted string. To do this:

1) Generate the ordered encryption key (use a loop) and call shuffle() on it.
2) Populate the player’s key with ‘\0’ (the NUL character).
3) Build the encrypted string:
    a. Allocate memory for it.
    b. Loop over the original string.
        i. If the current character is a letter, get the upper case, look it up in the encryption key and append to the encrypted string the encrypted value
        ii. Otherwise, output the original (this preserves spaces, commas, etc).
    c. NUL terminate the encrypted string

We will change updateState to accept a pair. If the input is not NULL and is not “quit”, then if it is 2 characters long, we will populate the player’s key with this new mapping. The first character is the distance, the second is the replacement. If the player entered something else, output a reasonable error message.

We will change displayWorld. It will output the encrypted string and the player’s progress:

Encrypted: QZZ ZQZ

Decrypted: ___ ___

The rule for decrypted:

if the encrypted character is A-Z,

look that character up in the player’s key. If the key is ‘\0’, output ‘_’,

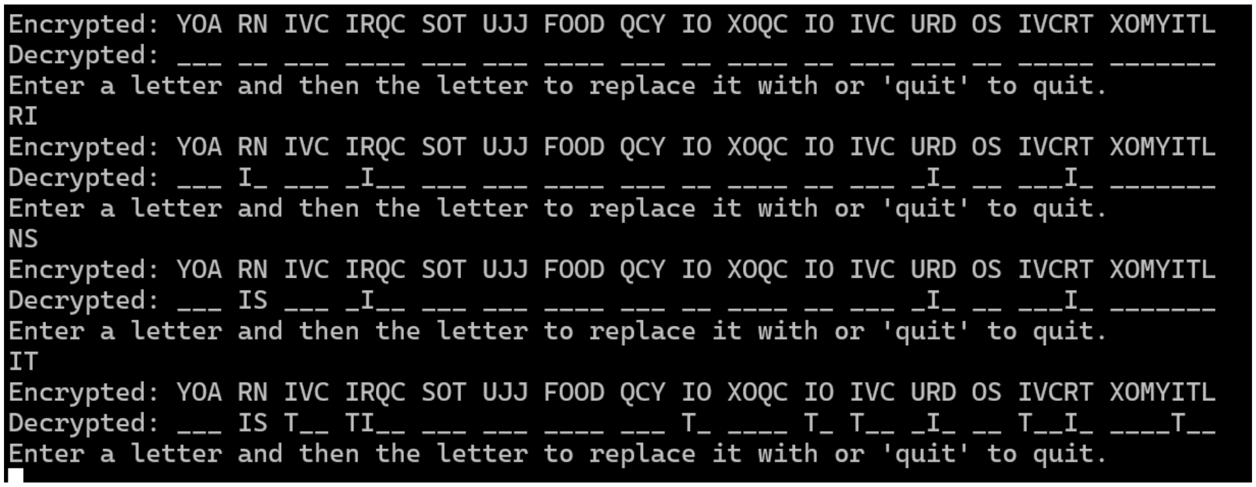
otherwise output the key.

Else

Output the encrypted character (spaces, punctuation, etc.)

Finally, in teardown, make sure that you free the memory that you allocated for the encrypted string.

With this work done, we can actually (sort of) play the game. We can enter pairs and see the decoded string. Here is a screenshot from my version:



```
Encrypted: YOA RN IVC IRQC SOT UJJ FOOD QCY IO XOQC IO IVC URD OS IVCRT XOMYITL
Decrypted: ___ __ ___ ____ ___ ___ ____ ___ __ ____ __ ___ ___ __ _____ _______
Enter a letter and then the letter to replace it with or 'quit' to quit.
RI
Encrypted: YOA RN IVC IRQC SOT UJJ FOOD QCY IO XOQC IO IVC URD OS IVCRT XOMYITL
Decrypted: ___ I_ ___ _I__ ___ ___ ____ ___ __ ____ __ ___ _I_ __ ___I_ _______
Enter a letter and then the letter to replace it with or 'quit' to quit.
NS
Encrypted: YOA RN IVC IRQC SOT UJJ FOOD QCY IO XOQC IO IVC URD OS IVCRT XOMYITL
Decrypted: ___ IS ___ _I__ ___ ___ ____ ___ __ ____ __ ___ _I_ __ ___I_ _______
Enter a letter and then the letter to replace it with or 'quit' to quit.
IT
Encrypted: YOA RN IVC IRQC SOT UJJ FOOD QCY IO XOQC IO IVC URD OS IVCRT XOMYITL
Decrypted: ___ IS T__ TI__ ___ ___ ____ ___ T_ ____ T_ T__ _I_ __ T__I_ ____T__
Enter a letter and then the letter to replace it with or 'quit' to quit.
```

Some useful C library functions you might need (you can look them up in "man"):

random() time() srandom()

strlen()  isalpha() toupper()

malloc() free()

## Your program <u>**must**</u> run through Valgrind without any memory leaks. Any memory leaks or negative Valgrind messages will result in significant penalty, regardless of the rubric below.

| Rubric | Poor | OK | Good | Great |
|---|---|---|---|---|
| Comments | None/Excessive (0) | "What" not "Why", few (5) | Some "what" comments or missing some (7) | Anything not obvious has reasoning (10) |
| Variable/Function naming | Single letters everywhere (0) | Lots of abbreviations (5) | Full words most of the time (8) | Full words, descriptive (10) |
| Structure | None of: indentation | | One of: indentation | Both of: indentation |

| | | | | |
|---|---|---|---|---|
| | matches braces {}, helper functions (0) | | matches braces {}, helper functions (5) | matches braces {}, helper functions (10) |
| Fisher-Yates | None(0) | | Significant attempt (5) | Correct(10) |
| Generate Encryption Map | None (0) | One of: Variable declared correctly, populated in a loop, shuffled (3) | Two of: Variable declared correctly, populated in a loop, shuffled (6) | All of: Variable declared correctly, populated in a loop, shuffled (10) |
| Player Map Generated | None (0) | One of: Variable declared correctly, populated in a loop (3) | | All of: Variable declared correctly, populated in a loop, (5) |
| Encrypted String | None (0) | Two of: Variable declared and allocated correctly, non-letters copied, letters encrypted, NUL terminated (5) | Three of: Variable declared and allocated correctly, non-letters copied, letters encrypted, NUL terminated (10) | All of: Variable declared and allocated correctly, non-letters copied, letters encrypted, NUL terminated (15) |
| UpdateState | No changes(0) | One of: accepts 2 character mappings, gives error if invalid input, populates playerMap (5) | Two of: accepts 2 character mappings, gives error if invalid input, populates playerMap (10) | All of: accepts 2 character mappings, gives error if invalid input, populates playerMap (15) |
| Display | No changes (0) | | | prints decryption in progress (10) |
| Teardown | No change(0) | | | Deallocates encrypted string (5) |