Name: KAMCHE YANN ARNAUD

Matricule: FE21A208

NB: Some Codes were too long to screenshot. I had to paste in the report

TASK: Propose an implementation of one of each category of substitution ciphers studied in class. Indicate the cipher chose in each category.

1) ***Simple Substitution Cipher: They replace each character of plaintext with a corresponding character of cipher text.***

Case Study: An implementation of CAESAR CIPHER in C

Code:

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

// Function declarations
void caesar_cipher(char* message, int key, int encrypt);

int main() {
    char message[100];
    int key;

    // Prompt the user for the message and key
    printf("Enter a message to encrypt: ");
    fgets(message, sizeof(message), stdin);
    printf("Enter a key value (1-25): ");
    scanf("%d", &key);

    // Encrypt the message
    caesar_cipher(message, key, 1);
    printf("Encrypted message: %s\n", message);

    // Decrypt the message
    caesar_cipher(message, key, 0);
    printf("Decrypted message: %s\n", message);

    return 0;
}

// Function to apply the Caesar Cipher to a message
void caesar_cipher(char* message, int key, int encrypt) {
    int len = strlen(message);
    for (int i = 0; i < len; i++) {
        if (isalpha(message[i])) {
            // Determine the base value for uppercase or lowercase letters
            char base = isupper(message[i]) ? 'A' : 'a';
            // Apply the Caesar Cipher shift
            char shifted = ((message[i] - base + (encrypt ? key : 26 - key)) % 26) + base;
            message[i] = shifted;
        }
    }
}
```

Screenshot

Key = 6

```
Enter a message to encrypt: I am using Caesar Cipher to encrypt this text
Enter a key value (1-25): 6
Encrypted message: O gs ayotm Igkygx Iovnkx zu ktixevz znoy zkdz
```

Key = 10

```
Enter a message to encrypt: I am using Caesar Cipher to encrypt this text
Enter a key value (1-25): 10
Encrypted message: S kw ecsxq Mkockb Mszrob dy oxmbizd drsc dohd
```

2) Polyalphabetic substitution cipher: They use of multiple mappings from plaintext to cipher text characters

Case Study: An Implementation of VIGINERE in C

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main()
{
    char inputString[100];
    char keyString [] = "SECURITY";
    int i, j;

    int keyLength = strlen(keyString);
    //Getting a value for inputString
    printf("Please enter a message: ");
    gets(inputString);
    int stringLength = strlen(inputString);
    char newKeyValue[stringLength], encryptedString[stringLength], decryptedString[stringLength];

    //Setting inputString to uppercase
    for(i=0; inputString[i]!='\0'; i++)
    {
    if(inputString[i]>='a' && inputString[i]<='z')
        {
            inputString[i] = inputString[i] - 32;
        }
    }
    //Creating new key according to inputString length
    for(i=0, j=0; i<stringLength; i++, j++)
    {
        if(j == keyLength)
            j=0;
            newKeyValue[i] = keyString[j];
    }
    newKeyValue[i] = '\0';
    //Encryption Section
    for(i=0; i< stringLength; i++)
        if(inputString[i] == 32) //Handling blank space between words
        {
            encryptedString[i] = 32; // assigning blank space to encrypted string
                }
                else
                {

        encryptedString[i] = ((inputString[i] + newKeyValue[i]) % 26) + 65;
}
    encryptedString[i] = '\0'; //Letting the code know that it reached the end of the string

    //Decryption Section
    for(i=0; i<stringLength; i++)
        if(encryptedString[i] == 32) //Handling blank space between word
        {
            decryptedString[i] = 32; // assigning blank space to decrypted string
        }
        else
        {

        decryptedString[i] = (((encryptedString[i] - newKeyValue[i]) + 26) % 26) + 65;
}
    decryptedString[i] = '\0'; //Letting the code know that it reached the end of the string
    //Output results
    printf("Plaintext: %s\n", inputString);
    printf("Keyword: %s\n", keyString);
    printf("Ciphertext: %s\n", encryptedString);
    printf("Decrypted Text: %s\n", decryptedString);
    return 0;
}
```

Screenshot

Key = “SECURTIY”

```
Please enter a message: This message is encrypted with Viginere Cipher
Plaintext: THIS MESSAGE IS ENCRYPTED WITH VIGINERE CIPHER
Keyword: SECURITY
Ciphertext: LLKM UXQKEIY QL WRELPXMCV YCKP TAKKHVZX UMRBVZ
Decrypted Text: THIS MESSAGE IS ENCRYPTED WITH VIGINERE CIPHER
```

Key = "ENCRYPTION"

```
Please enter a message: This message is encrypted with Viginere Cipher
Plaintext: THIS MESSAGE IS ENCRYPTED WITH VIGINERE CIPHER
Keyword: ENCRYPTION
Ciphertext: XUKJ BXAGNKR ZQ XVQECCVVB PQHU IKXGCXZS GVRYCG
Decrypted Text: THIS MESSAGE IS ENCRYPTED WITH VIGINERE CIPHER
```

3) ***Polygram Substitution ciphers: They are the most general, permitting arbitrary substitutions for groups of characters***

Case study: Implementation of Viginere Cipher in C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define KEY_SIZE 25
#define GRID_SIZE 5
void createKeyTable(char key[], char keyTable[GRID_SIZE][GRID_SIZE]) {

    int i, j, k;
    int len = strlen(key);
    int row = 0, col = 0;
    int used[26] = {0};
    for (k = 0; k < len; k++) {
        if (key[k] == 'j') {
            key[k] = 'i';
        }

        if (!isalpha(key[k])) {
            continue;
        }
        if (!used[key[k] - 'a']) {
            keyTable[row][col] = key[k];
            used[key[k] - 'a'] = 1;
```

```
            col++;

            if (col == GRID_SIZE) {
                col = 0;
                row++;
            }
        }
    }

    for (i = 0; i < 26; i++) {
        if (i == ('j' - 'a')) {
            continue;
        }

        if (!used[i]) {
            keyTable[row][col] = i + 'a';
            col++;

            if (col == GRID_SIZE) {
                col = 0;
                row++;
            }
        }
    }
}
void encrypt(char message[], char keyTable[GRID_SIZE][GRID_SIZE], char
encrypted[]) {
    int i, j, k;
    int len = strlen(message);
    int newLen = 0;
    char c1, c2;
    int row1, col1, row2, col2;


    for (i = 0; i < len; i++) {
        if (message[i] == 'j') {
```

```
            message[i] = 'i';
        }
    }


    for (i = 0; i < len; i += 2) {
        c1 = message[i];
        c2 = message[i + 1];
        row1 = col1 = row2 = col2 = -1;

        for (j = 0; j < GRID_SIZE; j++) {
            for (k = 0; k < GRID_SIZE; k++) {
                if (keyTable[j][k] == c1) {
                    row1 = j;
                    col1 = k;
                }

                if (keyTable[j][k] == c2) {
                    row2 = j;
                    col2 = k;
                }
            }
        }

        if (row1 == row2) { // Same row
            encrypted[newLen++] = keyTable[row1][(col1 + 1) % GRID_SIZE];
            encrypted[newLen++] = keyTable[row1][(col2 + 1) % GRID_SIZE];
        } else if (col1 == col2) { // Same column
            encrypted[newLen++] = keyTable[(row1 + 1) % GRID_SIZE][col1];
            encrypted[newLen++] = keyTable[(row2 + 1) % GRID_SIZE][col1];
        } else { // Form a rectangle
            encrypted[newLen++] = keyTable[row1][col2];
            encrypted[newLen++] = keyTable[row2][col1];
        }
    }
```

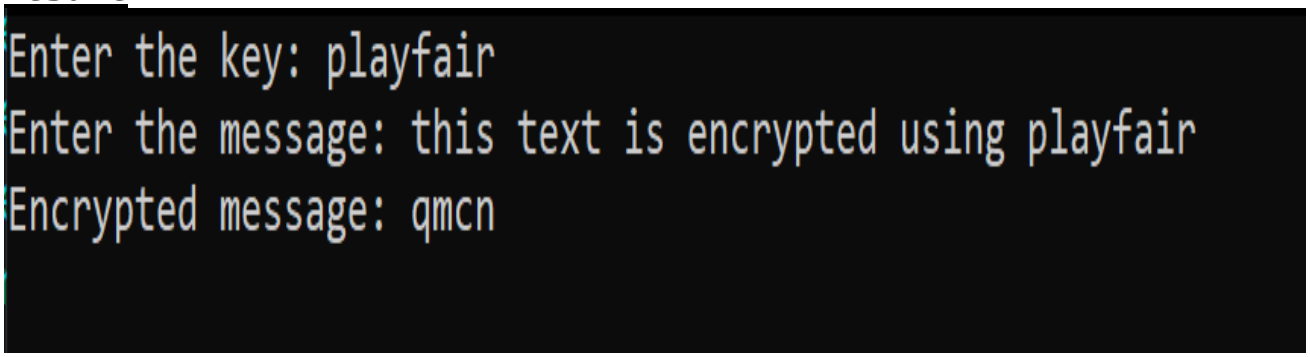
```
    encrypted[newLen] = '\0';
}

int main() {
    char key[KEY_SIZE] = {'\0'};
    char keyTable[GRID_SIZE][GRID_SIZE] = {{'\0'}};
    char message[KEY_SIZE] = {'\0'};
    char encrypted[KEY_SIZE * 2] = {'\0'};

    printf("Enter the key: ");
    fgets(key, KEY_SIZE, stdin);
    key[strcspn(key, "\n")] = '\0';
    printf("Enter the message: ");
    fgets(message, KEY_SIZE, stdin);
    message[strcspn(message, "\n")] = '\0';
    createKeyTable(key, keyTable);
    encrypt(message, keyTable, encrypted);
    printf("Encrypted message: %s\n", encrypted);

    return 0;
}
```

Result



4) Polyalphabetic Substitution ciphers: They use multiple mappings from plaintext

Case Study: Implementation of Beale Cipher in C

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX_WORDS 1000

#define MAX_DIGITS 10

typedef struct {

    int page;

    int line;

    int word;

} Ciphertext;


int read_ciphertext(char *filename, Ciphertext *ciphertexts, int
num_ciphertexts);

void decode_ciphertext(char *book_filename, Ciphertext *ciphertexts, int
num_ciphertexts);


int main(int argc, char *argv[]) {

    if (argc != 3) {

        printf("Usage: %s <ciphertext_file> <book_file>\n", argv[0]);

        return 1;

    }


    char *ciphertext_filename = argv[1];

    char *book_filename = argv[2];


    Ciphertext ciphertexts[3];

    int num_ciphertexts = read_ciphertext(ciphertext_filename, ciphertexts,
3);


    if (num_ciphertexts != 3) {

        printf("Error: Expected 3 ciphertexts, but found %d\n",
num_ciphertexts);

        return 1;

    }
```

```
    decode_ciphertext(book_filename, ciphertexts, 3);


    return 0;

}


int read_ciphertext(char *filename, Ciphertext *ciphertexts, int
num_ciphertexts) {

    FILE *fp = fopen(filename, "r");

    if (fp == NULL) {

        printf("Error: Could not open file '%s'\n", filename);

        return 0;

    }


    int i;

    for (i = 0; i < num_ciphertexts; i++) {

        if (fscanf(fp, "%d,%d,%d", &ciphertexts[i].page,
&ciphertexts[i].line, &ciphertexts[i].word) != 3) {

            printf("Error: Invalid ciphertext in file '%s'\n", filename);

            fclose(fp);

            return i;

        }

    }


    fclose(fp);

    return i;

}


void decode_ciphertext(char *book_filename, Ciphertext *ciphertexts, int
num_ciphertexts) {

    FILE *fp = fopen(book_filename, "r");

    if (fp == NULL) {

        printf("Error: Could not open file '%s'\n", book_filename);

        return;

    }


    char word[MAX_DIGITS + 1];

    int page = 0;
```

```
int line = 0;
int word_num = 0;
int i;

while (fgets(word, MAX_DIGITS + 1, fp) != NULL) {
    int len = strlen(word);
    if (len > 0 && word[len-1] == '\n') {
        word[len-1] = '\0';
    }

    word_num++;

    if (word_num > MAX_WORDS) {
        printf("Error: Book file contains too many words\n");
        fclose(fp);
        return;
    }

    if (strcmp(word, "PAGE") == 0) {
        if (fscanf(fp, "%d", &page) != 1) {
            printf("Error: Invalid book file format\n");
            fclose(fp);
            return;
        }
        line = 0;
        word_num = 0;
    } else if (strcmp(word, "LINE") == 0) {
        if (fscanf(fp, "%d", &line) != 1) {
            printf("Error: Invalid book file format\n");
            fclose(fp);
            return;
        }
        word_num = 0;
    } else {
        for (i = 0; i < num_ciphertexts; i++) {
```

```
                if (page == ciphertexts[i].page && line ==
ciphertexts[i].line && word_num == ciphertexts[i].word) {
                    printf("%d ", i);
                    break;
                }
            }
        }
    }

    fclose(fp);
}
```

Result: