**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.

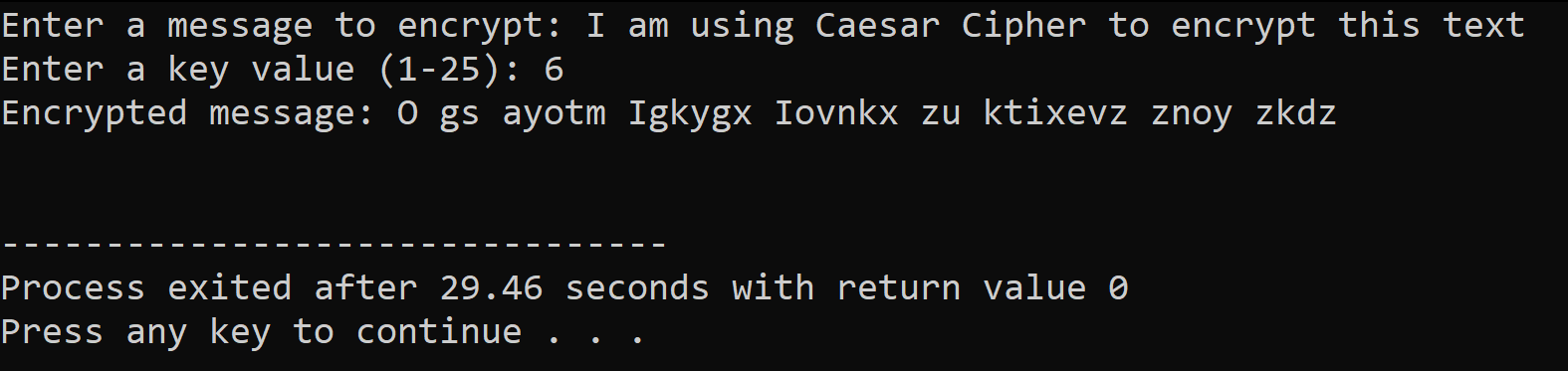
1. ***Simple Substitution Cipher: They replace each character of plaintext with a corresponding character of cipher text.***

**Case Study: An implementation of CAESAR CIPHER in C**

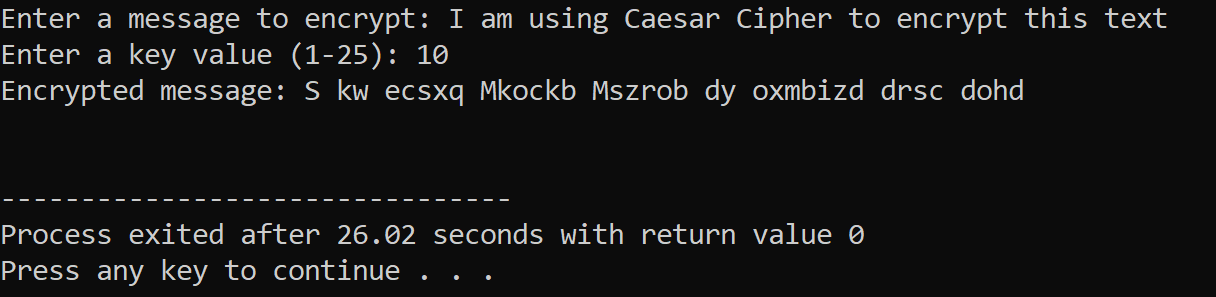
**Code:**



**Screenshot  
Key = 6**



**Key = 10**

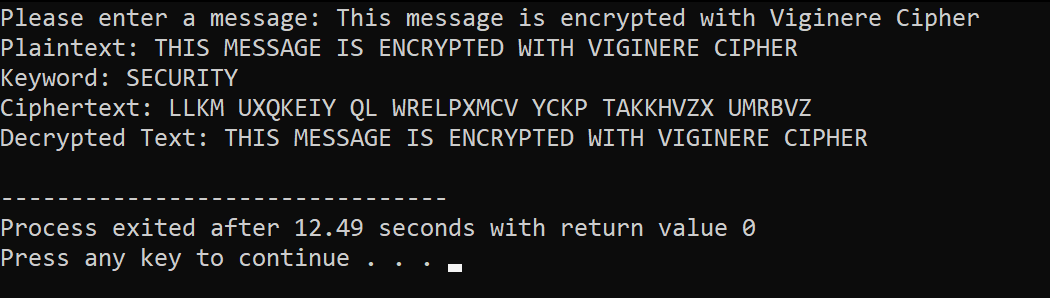


1. ***Polyalphabetic substitution cipher: They use of multiple mappings from plaintext to cipher text characters***

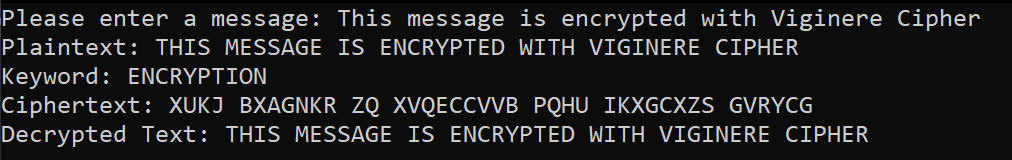
**Case Study: An Implementation of VIGINERE in C**



**Screenshot**  
**Key = “SECURTIY”**



**Key = “ENCRYPTION”**



1. ***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';

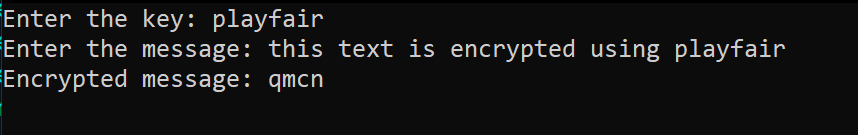
createKeyTable(key, keyTable);

encrypt(message, keyTable, encrypted);

printf("Encrypted message: %s\n", encrypted);

return 0;

}

**Result**

1. ***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:

