```
/*Assignment: pa01 - Encrypting a plaintext file using the Hill Cypher

Author: Ruben Bernard
Language: C
To compile: gcc -o pa01 pa01.c
To execute: -> ./pa01 kX.txt pX.txt
Class: CIS3360 - SECURITY IN COMPUTING - FALL 2024
Instructor: McAlpin
Due date: 9-30-2024
*/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>

// read matrix text from the key file
// opens files, reads file and matrix size
int** readFile(char *filename, int *m){
    FILE *key = fopen(filename, "r"); // open file
    if (key == NULL) {

        exit(1); // exit on error
    }
    fscanf(key, "%d", m); // scans file for matrix size

    int **matrix = malloc(sizeof(int*) * *m); // allocates memory for the matrix
array
    for (int i = 0; i < *m; i++) {
        matrix[i] = malloc(*m * sizeof(int)); // allocate memory for each row
    }
    for (int i = 0; i < *m; i++) {
        for (int j = 0; j < *m; j++) { // 2D like McAlpin recommended
            fscanf(key, "%d", &matrix[i][j]); // scans for column and row
        }
    }
    fclose(key); // closes file pointer
    return matrix; // returns matrix, type integer
}

char* rPlain(char *filename, int n, int *tLength) { // reads plaintext file
    FILE *plainTextFile = fopen(filename, "r"); // opens and reads
    if (plainTextFile == NULL) {
        exit(1); // exit on error
    }

    char *plainText = malloc(10000 * sizeof(char)); // allocates memory for
character, times 10000 like rubric said
    if (plainText == NULL) {
        printf("Memory allocation failed.\n");
        fclose(plainTextFile);
        exit(1); // exit on error
    }

    int index = 0;
    char ch;

    while ((ch = fgetc(plainTextFile)) != EOF) { // end of file, proceeds if not at
end
```

```
        if (isalpha(ch)) { // checks if alphabetic letter
            plainText[index++] = tolower(ch); // convert to lowercase
        }
    }

    while (index % n != 0) { // pads with 'x' if length is not divisible by n
        plainText[index++] = 'x'; // padding with x
    }

    *tLength = index; // sets length to index
    fclose(plainTextFile); // closes file
    return plainText;
}

void matrixVector(int **matrix, int *vector, int *result, int n) { // vector for
matrix, 4 parameters
    for (int i = 0; i < n; i++) {
        result[i] = 0; // initialize result to 0
        for (int j = 0; j < n; j++) {
            result[i] += matrix[i][j] * vector[j]; // multiplies matrix element and
vector
        }
        result[i] %= 26; // modulo by 26 (letters in alphabet)
    }
}

char* textencryption(int **key, char *plaintext, int n, int length) {
    char *ciphertext = malloc(length * sizeof(char)); // allocate memory for
ciphertext
    int vector[n]; // sets vector size to n
    int result[n];

    for (int i = 0; i < length; i += n) {
        for (int j = 0; j < n; j++) {
            vector[j] = plaintext[i + j] - 'a'; // convert plaintext character to
numeric value
        }
        matrixVector(key, vector, result, n); // passes matrixVector function
through

        for (int j = 0; j < n; j++) { // variable j instead of r
            ciphertext[i + j] = result[j] + 'a'; // sets ciphertext to final
variable plus 'a'
        }
    }
    return ciphertext;
}

void pCipher(char *ciphertext, int length) {
    for (int i = 0; i < length; i++) { // plaintext cipher loops and prints output
        printf("%c", ciphertext[i]); //

        if ((i + 1) % 80 == 0) { // newline if over 80 characters
            printf("\n");
        }
    }
    printf("\n");
}
```

```
int main(int argc, char *argv[]) { // like the command video McAlpin showed
    if (argc != 3) {
        printf("Usage: %s keyfile plaintextfile\n", argv[0]); // usage message
        return 1;
    }

    int m;
    int **key = readFile(argv[1], &m); // read key matrix

    int length;
    char *plaintext = rPlain(argv[2], m, &length); // read plaintext

    printf("\nKey matrix:\n");
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            printf("%4d", key[i][j]); // %4d to align the matrix like the rubric
says
        }
        printf("\n");
    }

    printf("\nPlaintext:\n");
    for (int i = 0; i < length; i++) {
        printf("%c", plaintext[i]);
        if ((i + 1) % 80 == 0) { // prints a newline if 80 characters
            printf("\n");
        }
    }
    printf("\n");

    char *cipher = textencryption(key, plaintext, m, length); // encrypt the
plaintext

    printf("\nCiphertext:\n");
    pCipher(cipher, length); // print ciphertext

    for (int i = 0; i < m; i++) {
        free(key[i]); // free each row
    }
    free(key); // free key matrix
    free(plaintext); // free plaintext
    free(cipher); // free ciphertext
    return 0;
}

/*============================================================================
| I RUBEN BERNARD (ru692104) affirm that this program is
| entirely my own work and that I have neither developed my code together with
| any another person, nor copied any code from any other person, nor permitted
| my code to be copied or otherwise used by any other person, nor have I
| copied, modified, or otherwise used programs created by others. I acknowledge
| that any violation of the above terms will be treated as academic dishonesty.
+===========================================================================*/
```