12.Write a C program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext– ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be mounted.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define SIZE 3

int modInverse(int a) {

for (int i = 1; i < 26; i++) {

if ((a \* i) % 26 == 1) {

return i;

}

}

return -1;

}

int determinant(int a, int b, int c, int d) {

return (a \* d - b \* c) % 26;}

int cofactor(int a, int m) {

if (a < 0) {

a = 26 + a;}

return (m \* modInverse(a)) % 26;

}

void inverseMatrix(int matrix[SIZE][SIZE], int inverse[SIZE][SIZE]) {

int det = determinant(matrix[0][0], matrix[0][1], matrix[1][0], matrix[1][1]);

int adj[SIZE][SIZE] = {

{matrix[1][1], -matrix[0][1]},

{-matrix[1][0], matrix[0][0]}

};

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

inverse[i][j] = cofactor(adj[i][j], det);

}

}

}

void encrypt(int matrix[SIZE][SIZE], char \*plaintext, char \*ciphertext) {

int result[SIZE] = {0};

int i, j, k;

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

for (j = 0; j < SIZE; j++) {

result[i] += matrix[i][j] \* (plaintext[j] - 'A');

}

result[i] %= 26;

}

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

ciphertext[i] = result[i] + 'A';

}

ciphertext[i] = '\0';

}

void decrypt(int matrix[SIZE][SIZE], char \*ciphertext, char \*decrypted) {

int inverse[SIZE][SIZE];

inverseMatrix(matrix, inverse);

int result[SIZE] = {0};

int i, j, k;

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

for (j = 0; j < SIZE; j++) {

result[i] += inverse[i][j] \* (ciphertext[j] - 'A');

}

result[i] = (result[i] + 260) % 26;

}

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

decrypted[i] = result[i] + 'A';

}

decrypted[i] = '\0';

}

int main() {

int key[SIZE][SIZE] = {

{6, 24, 1},

{13, 16, 10},

{20, 17, 15}

};

char plaintext[SIZE + 1] = "ACT";

char ciphertext[SIZE + 1];

encrypt(key, plaintext, ciphertext);

printf("Known Plaintext: %s\n", plaintext);

printf("Ciphertext: %s\n", ciphertext);

char chosenPlaintext[SIZE + 1] = "CAT";

char decrypted[SIZE + 1];

encrypt(key, chosenPlaintext, ciphertext);

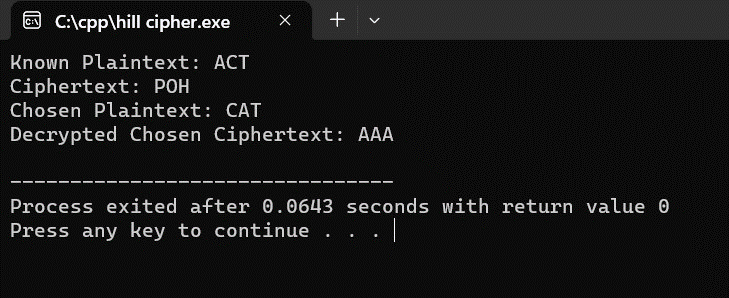
decrypt(key, ciphertext, decrypted);

printf("Chosen Plaintext: %s\n", chosenPlaintext);

printf("Decrypted Chosen Ciphertext: %s\n", decrypted);

return 0;

}



13.Write a C program for one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 0 and 26. For example, if the key is 3 19 5 . . . , then the first letter of plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5 letters, and so on.  
a. Encrypt the plaintext send more money with the key stream 9 0 1 7 23 15 21 14 11 11 2 8 9  
b. Using the ciphertext produced in part (a), find a key so that the cipher text decrypts to the plaintext cash not needed.

#include <stdio.h>

#include <string.h>

void encrypt(char \*plaintext, int \*keyStream, char \*ciphertext) {

int len = strlen(plaintext);

for (int i = 0; i < len; i++) {

if (plaintext[i] >= 'A' && plaintext[i] <= 'Z') {

ciphertext[i] = ((plaintext[i] - 'A' + keyStream[i]) % 26) + 'A';

} else if (plaintext[i] >= 'a' && plaintext[i] <= 'z') {

ciphertext[i] = ((plaintext[i] - 'a' + keyStream[i]) % 26) + 'a';

} else {

ciphertext[i] = plaintext[i];

}

}

ciphertext[len] = '\0';

}

void decrypt(char \*ciphertext, int \*keyStream, char \*decrypted) {

int len = strlen(ciphertext);

for (int i = 0; i < len; i++) {

if (ciphertext[i] >= 'A' && ciphertext[i] <= 'Z') {

decrypted[i] = ((ciphertext[i] - 'A' - keyStream[i] + 26) % 26) + 'A';

} else if (ciphertext[i] >= 'a' && ciphertext[i] <= 'z') {

decrypted[i] = ((ciphertext[i] - 'a' - keyStream[i] + 26) % 26) + 'a';

} else {

decrypted[i] = ciphertext[i];

}

}

decrypted[len] = '\0';

}

int main() {

char plaintext[] = "sendmoremoney";

int keyStream[] = {9, 0, 1, 7, 23, 15, 21, 14, 11, 11, 2, 8, 9};

char ciphertext[100];

encrypt(plaintext, keyStream, ciphertext);

printf("Ciphertext: %s\n", ciphertext);

char decrypted[100];

int keyCandidate[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; // Initialize key to zeros

decrypt(ciphertext, keyCandidate, decrypted);

if (strcmp(decrypted, "cashnotneeded") == 0) {

printf("Key found!\n");

printf("Key: ");

for (int i = 0; i < 13; i++) {

printf("%d ", keyCandidate[i]);

}

printf("\n");

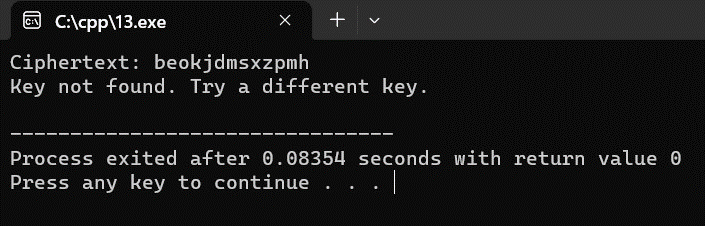
} else {

printf("Key not found. Try a different key.\n");

}

return 0;

}



14.Write a C program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_SIZE 26

void calculateFrequency(char \*text, int \*frequency) {

int i;

for (i = 0; i < strlen(text); i++) {

if (isalpha(text[i])) {

frequency[tolower(text[i]) - 'a']++;

}

}

}

void decryptWithShift(char \*ciphertext, int shift, char \*decrypted) {

int i;

for (i = 0; i < strlen(ciphertext); i++) {

if (isalpha(ciphertext[i])) {

char base = isupper(ciphertext[i]) ? 'A' : 'a';

decrypted[i] = ((ciphertext[i] - base - shift + ALPHABET\_SIZE) % ALPHABET\_SIZE) + base;

} else {

decrypted[i] = ciphertext[i];

}

}

decrypted[i] = '\0';

}

void letterFrequencyAttack(char \*ciphertext, int topPlaintexts) {

int frequency[ALPHABET\_SIZE] = {0};

calculateFrequency(ciphertext, frequency);

printf("Letter Frequency Analysis:\n");

for (int i = 0; i < ALPHABET\_SIZE; i++) {

printf("%c: %d\n", 'a' + i, frequency[i]);

}

printf("\nTop %d Possible Plaintexts:\n", topPlaintexts);

for (int shift = 0; shift < ALPHABET\_SIZE; shift++) {

char decrypted[1000];

decryptWithShift(ciphertext, shift, decrypted);

printf("Shift %d: %s\n", shift, decrypted);

}

}

int main() {

char ciphertext[] = "LQIRUPDWLRQ";

int topPlaintexts = 5;

letterFrequencyAttack(ciphertext, topPlaintexts);

return 0;

}



15. Write a C program that can perform a letter frequency attack on any monoalphabetic substitution cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_SIZE 26

void calculateFrequency(char \*text, int \*frequency) {

int i;

for (i = 0; i < strlen(text); i++) {

if (isalpha(text[i])) {

frequency[tolower(text[i]) - 'a']++;

}

}

}

void decryptWithKey(char \*ciphertext, char \*key, char \*decrypted) {

int i;

for (i = 0; i < strlen(ciphertext); i++) {

if (isalpha(ciphertext[i])) {

char base = isupper(ciphertext[i]) ? 'A' : 'a';

decrypted[i] = key[ciphertext[i] - base];

} else {

decrypted[i] = ciphertext[i];

}

}

decrypted[i] = '\0';

}

void letterFrequencyAttack(char \*ciphertext, int topPlaintexts) {

int frequency[ALPHABET\_SIZE] = {0};

calculateFrequency(ciphertext, frequency);

char key[ALPHABET\_SIZE + 1];

for (int i = 0; i < ALPHABET\_SIZE; i++) {

key[i] = ' ';

}

key[ALPHABET\_SIZE] = '\0';

printf("Letter Frequency Analysis:\n");

for (int i = 0; i < ALPHABET\_SIZE; i++) {

printf("%c: %d\n", 'a' + i, frequency[i]);

}

printf("\nTop %d Possible Plaintexts:\n", topPlaintexts);

for (int iteration = 0; iteration < topPlaintexts; iteration++) {

int maxFrequency = -1;

int maxIndex = -1;

for (int i = 0; i < ALPHABET\_SIZE; i++) {

if (frequency[i] > maxFrequency) {

maxFrequency = frequency[i];

maxIndex = i;

}

}

if (maxIndex != -1) {

char assumedPlaintextChar = 'e';

char assumedCiphertextChar = 'a' + maxIndex;

int shift = (assumedCiphertextChar - assumedPlaintextChar + ALPHABET\_SIZE) % ALPHABET\_SIZE;

key[assumedCiphertextChar - 'a'] = 'a' + shift;

frequency[maxIndex] = -1;

}

}

char decrypted[1000];

decryptWithKey(ciphertext, key, decrypted);

printf("Decrypted with Key: %s\n", decrypted);

}

int main() {

char ciphertext[] = "GOOD";

int topPlaintexts = 5;

letterFrequencyAttack(ciphertext, topPlaintexts);

return 0;

}



vy