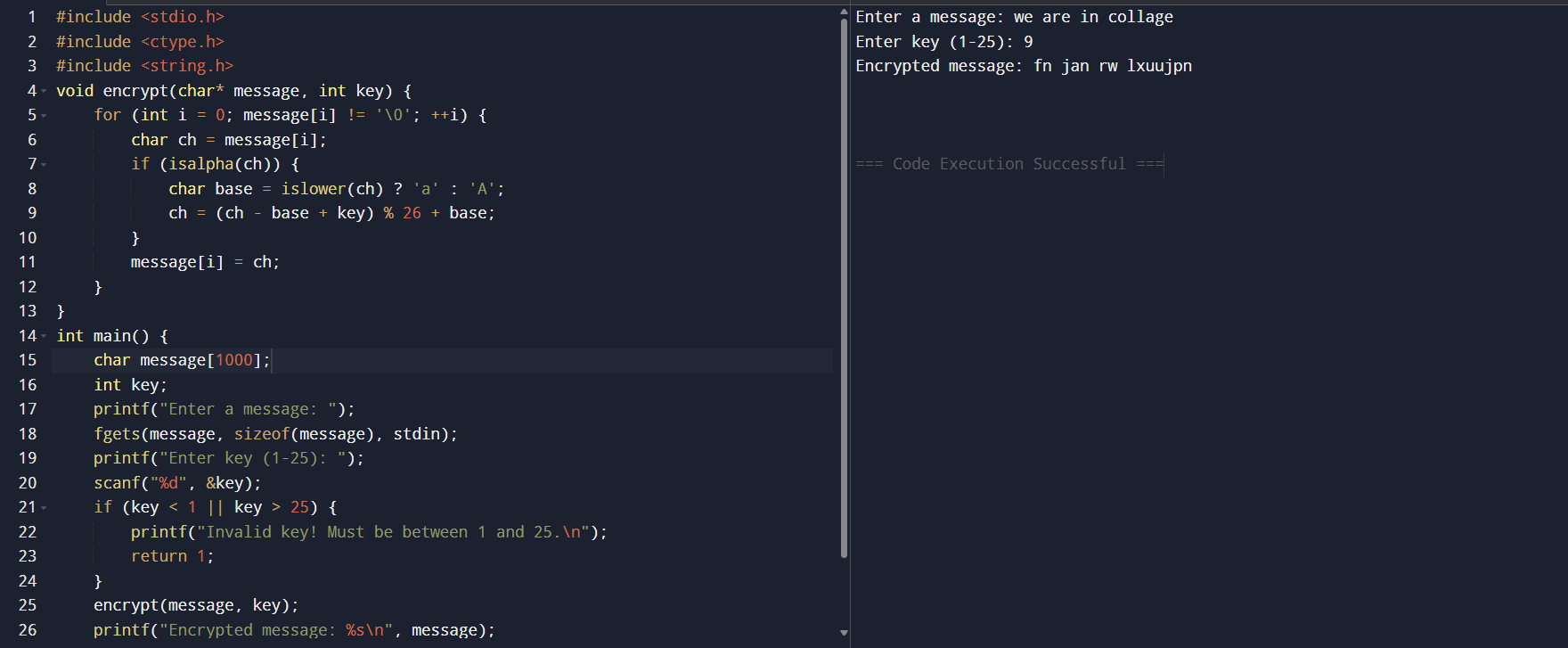
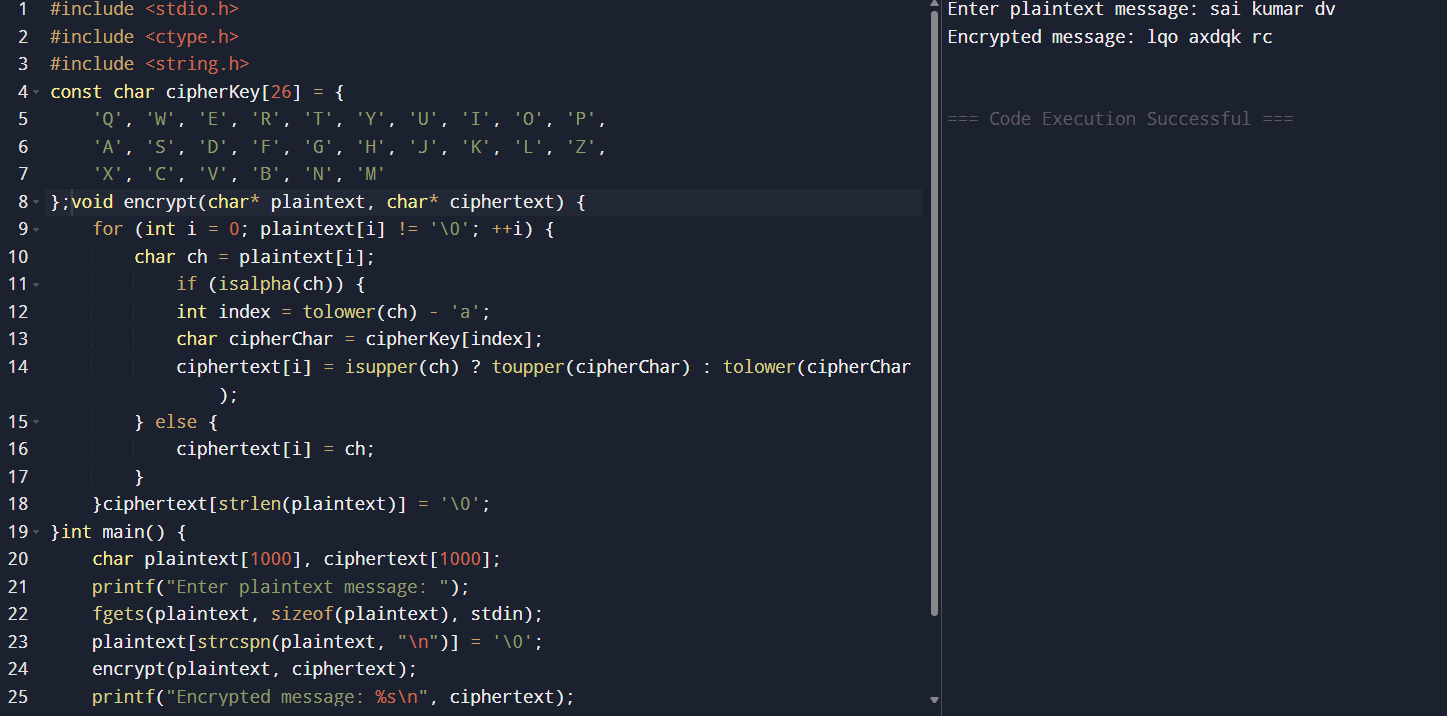
D V SAI KUMAR(192372321)

LIST OF PROGRAMS

1.Write a C program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25.



2. Write a C program for monoalphabetic substitution cipher maps a plaintext alphabet to a ciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.



3. Write a C program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

char matrix[SIZE][SIZE];

void generateMatrix(char\* key) {

int alpha[26] = {0};

int i, j, k = 0;

alpha['j' - 'a'] = 1;

for (i = 0; key[i]; i++) {

char ch = tolower(key[i]);

if (ch == 'j') ch = 'i';

if (isalpha(ch) && !alpha[ch - 'a']) {

matrix[k / SIZE][k % SIZE] = ch;

alpha[ch - 'a'] = 1;

k++;

}

}

for (i = 0; i < 26; i++) {

if (!alpha[i]) {

matrix[k / SIZE][k % SIZE] = i + 'a';

k++;

}

}

}

void prepareText(char\* input, char\* output) {

int i, j = 0;

for (i = 0; input[i]; i++) {

if (isalpha(input[i])) {

char ch = tolower(input[i]);

output[j++] = (ch == 'j') ? 'i' : ch;

}

}

output[j] = '\0';

for (i = 0; i < j; i += 2) {

if (output[i] == output[i + 1]) {

memmove(output + i + 2, output + i + 1, strlen(output) - i);

output[i + 1] = 'x';

j++;

}

}

if (j % 2 != 0) {

output[j++] = 'x';

output[j] = '\0';

}

}

void findPosition(char ch, int\* row, int\* col) {

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

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

if (matrix[i][j] == ch) {

\*row = i;

\*col = j;

return;

}

}

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

int i;

for (i = 0; plaintext[i]; i += 2) {

int r1, c1, r2, c2;

findPosition(plaintext[i], &r1, &c1);

findPosition(plaintext[i + 1], &r2, &c2);

if (r1 == r2) {

ciphertext[i] = matrix[r1][(c1 + 1) % SIZE];

ciphertext[i + 1] = matrix[r2][(c2 + 1) % SIZE];

} else if (c1 == c2) {

ciphertext[i] = matrix[(r1 + 1) % SIZE][c1];

ciphertext[i + 1] = matrix[(r2 + 1) % SIZE][c2];

} else {

ciphertext[i] = matrix[r1][c2];

ciphertext[i + 1] = matrix[r2][c1];

}

}

ciphertext[i] = '\0';

}MONARCHY

void printMatrix() {

printf("Playfair Matrix:\n");

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

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

printf("%c ", matrix[i][j]);

}

printf("\n");

}

}

int main() {

char key[100], plaintext[1000], formatted[1000], ciphertext[1000];

printf("Enter keyword: ");

scanf("%s", key);

printf("Enter plaintext: ");

scanf(" %[^\n]", plaintext);

generateMatrix(key);

printMatrix();

prepareText(plaintext, formatted);

encrypt(formatted, ciphertext);

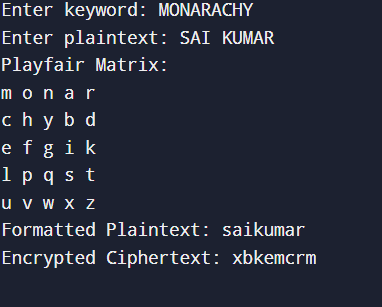
printf("Formatted Plaintext: %s\n", formatted);

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

return 0;

}

OUTPUT:



4. Write a C program for polyalphabetic substitution cipher uses a separate monoalphabetic substitution cipher for each successive letter of plaintext, depending on a key.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void generateKey(char\* key, int msgLen, char\* newKey) {

int keyLen = strlen(key);

for (int i = 0, j = 0; i < msgLen; i++) {

if (isalpha(key[j])) {

newKey[i] = tolower(key[j % keyLen]);

j++;

} else {

i--;

j++;

}

}

newKey[msgLen] = '\0';

}

void encrypt(char\* plaintext, char\* key, char\* ciphertext) {

int len = strlen(plaintext);

char newKey[len];

generateKey(key, len, newKey);

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

char ptChar = plaintext[i];

if (isalpha(ptChar)) {

char base = islower(ptChar) ? 'a' : 'A';

char shift = tolower(newKey[i]) - 'a';

ciphertext[i] = ((ptChar - base + shift) % 26) + base;

} else {

ciphertext[i] = ptChar;

}

}

ciphertext[len] = '\0';

}

int main() {

char plaintext[1000], key[100], ciphertext[1000];

printf("Enter the plaintext: ");

fgets(plaintext, sizeof(plaintext), stdin);

plaintext[strcspn(plaintext, "\n")] = '\0';

printf("Enter the key: ");

scanf("%s", key);

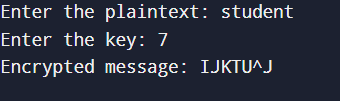
encrypt(plaintext, key, ciphertext);

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

return 0;

}

OUTPUT:



5. Write a C program for generalization of the Caesar cipher, known as the affine Caesar cipher, has the following form: For each plaintext letter p, substitute the ciphertext letter C: C = E([a, b], p) = (ap + b) mod 26 A basic requirement of any encryption algorithm is that it be one-to-one. That is, if p q, then E(k, p) E(k, q). Otherwise, decryption is impossible, because more than one plaintext character maps into the same ciphertext character. The affine Caesar cipher is not one-to-one for all values of a. For example, for a = 2 and b = 3, then E([a, b], 0) = E([a, b], 13) = 3. a. Are there any limitations on the value of b? b. Determine which values of a are not allowed.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int gcd(int a, int b) {

while (b != 0) {

int t = b;

b = a % b;

a = t;

}

return a;

}

int modInverse(int a) {

a = a % 26;

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

if ((a \* x) % 26 == 1)

return x;

}

return -1;

}

char affineEncryptChar(char ch, int a, int b) {

if (isalpha(ch)) {

ch = toupper(ch);

int x = ch - 'A';

int e = (a \* x + b) % 26;

return e + 'A';

}

return ch;

}

char affineDecryptChar(char ch, int a, int b) {

if (isalpha(ch)) {

ch = toupper(ch);

int a\_inv = modInverse(a);

if (a\_inv == -1) return '?';

int y = ch - 'A';

int d = (a\_inv \* (y - b + 26)) % 26;

return d + 'A';

}

return ch;

}

int main() {

char plaintext[1000], ciphertext[1000], decrypted[1000];

int a, b;

printf("Enter values for 'a' and 'b' (a must be coprime to 26): ");

scanf("%d %d", &a, &b);

if (gcd(a, 26) != 1) {

printf("Invalid value for 'a'. It must be coprime to 26.\n");

return 1;

}

printf("Enter the plaintext (letters only): ");

scanf(" %[^\n]", plaintext);

for (int i = 0; i < strlen(plaintext); i++)

ciphertext[i] = affineEncryptChar(plaintext[i], a, b);

ciphertext[strlen(plaintext)] = '\0';

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

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

decrypted[i] = affineDecryptChar(ciphertext[i], a, b);

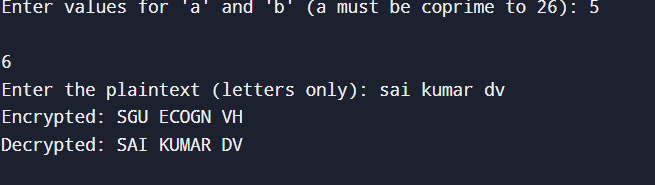
decrypted[strlen(ciphertext)] = '\0';

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

return 0;

}

OUTPUT:



6. Write a C program for ciphertext has been generated with an affine cipher. The most frequent letter of the ciphertext is “B,” and the second most frequent letter of the ciphertext is “U.”Break this code.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int modInverse(int a) {

a = a % 26;

for (int x = 1; x < 26; x++)

if ((a \* x) % 26 == 1)

return x;

return -1;

}

char decryptChar(char ch, int a, int b) {

if (!isalpha(ch)) return ch;

ch = toupper(ch);

int y = ch - 'A';

int a\_inv = modInverse(a);

if (a\_inv == -1) return '?';

int p = (a\_inv \* (y - b + 26)) % 26;

return p + 'A';

}

int solveKey(int c1, int c2, int p1, int p2, int\* a, int\* b) {

int diffP = (p1 - p2 + 26) % 26;

int diffC = (c1 - c2 + 26) % 26;

int inv = modInverse(diffP);

if (inv == -1) return 0;

\*a = (diffC \* inv) % 26;

\*b = (c1 - (\*a \* p1 % 26) + 26) % 26;

return 1;

}

int main() {

char ciphertext[1000] = "BUBUBUBUBUBUBUBUBUBU";

char plaintext[1000];

int a, b;

int p1 = 4, c1 = 1;

int p2 = 19, c2 = 20;

if (!solveKey(c1, c2, p1, p2, &a, &b)) {

printf("Unable to find valid 'a'. Decryption not possible.\n");

return 1;

}

printf("Recovered Key: a = %d, b = %d\n", a, b);

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

plaintext[i] = decryptChar(ciphertext[i], a, b);

}

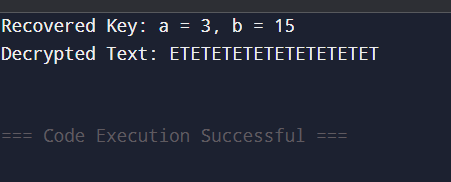
plaintext[strlen(ciphertext)] = '\0';

printf("Decrypted Text: %s\n", plaintext);

return 0;

}

OUTPUT:



7. Write a C program for the following ciphertext was generated using a simple substitution algorithm. 53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:‡\*8†83 (88)5\*†;46(;88\*96\*?;8)\*‡(;485);5\*†2:\*‡(;4956\*2(5\*—4)8¶8\* ;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806\*81 (‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;

#include <stdio.h>

#include <string.h>

char substitute(char symbol) {

switch(symbol) {

case '†': return 'E';

case '‡': return 'T';

case '(': return 'H';

case ')': return 'E';

case '\*': return 'S';

case ';': return 'O';

case '8': return 'N';

case '4': return 'A';

case '5': return 'R';

case '6': return 'D';

case '2': return 'L';

case '3': return 'M';

case '0': return 'C';

case '1': return 'U';

case '9': return 'I';

case '—': return 'Y';

case ':': return 'G';

case ']': return 'B';

case '?': return 'F';

case '.': return 'P';

case '¶': return 'K';

default: return symbol;

}

}

int main() {

char ciphertext[] =

"53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:"

"‡\*8†83(88)5\*†;46(;88\*96\*?;8)\*‡(;485);5\*†2:\*‡(;4956\*2"

"(5\*—4)8¶8\*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;"

"4)485†528806\*81(‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;";

char plaintext[1000];

int i;

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

plaintext[i] = substitute(ciphertext[i]);

}

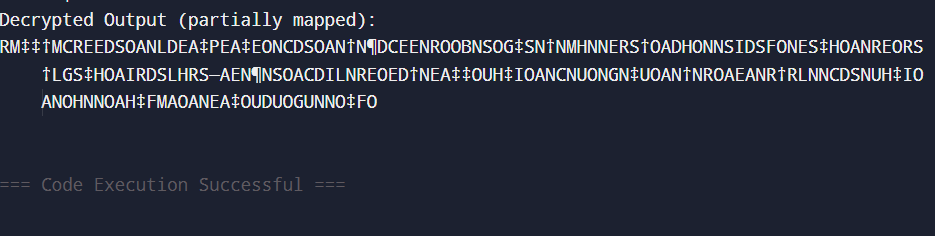
plaintext[i] = '\0';

printf("Decrypted Output (partially mapped):\n%s\n", plaintext);

return 0;

}

OUTPUT:



8. Write a C program for monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated.

For example, using the keyword CIPHER, write out the keyword followed by unused letters in normal order and match this against the plaintext letters: plain: a b c d e f g h i j k l m n o p q r s t u v w x y z cipher: C I P H E R A B D F G J K L M N O Q S T U V W X Y Z

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_LEN 26

void generateCipherAlphabet(char keyword[], char cipher[]) {

int used[26] = {0};

int i, j = 0;

for (i = 0; keyword[i] != '\0'; i++) {

char ch = toupper(keyword[i]);

if (ch >= 'A' && ch <= 'Z' && !used[ch - 'A']) {

cipher[j++] = ch;

used[ch - 'A'] = 1;

}

}

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

if (!used[i]) {

cipher[j++] = 'A' + i;

}

}

cipher[j] = '\0';

}

void encrypt(char plaintext[], char cipher[], char ciphertext[]) {

for (int i = 0; plaintext[i] != '\0'; i++) {

char ch = tolower(plaintext[i]);

if (ch >= 'a' && ch <= 'z') {

ciphertext[i] = cipher[ch - 'a'];

} else {

ciphertext[i] = plaintext[i];

}

}

ciphertext[strlen(plaintext)] = '\0';

}

int main() {

char keyword[] = "CIPHER";

char cipher[ALPHABET\_LEN + 1];

char plaintext[1000], ciphertext[1000];

generateCipherAlphabet(keyword, cipher);

printf("Generated Cipher Alphabet:\n");

printf("Plain : ");

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

printf("%c ", 'A' + i);

}

printf("\nCipher: ");

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

printf("%c ", cipher[i]);

}

printf("\n\nEnter plaintext: ");

fgets(plaintext, sizeof(plaintext), stdin);

plaintext[strcspn(plaintext, "\n")] = 0;

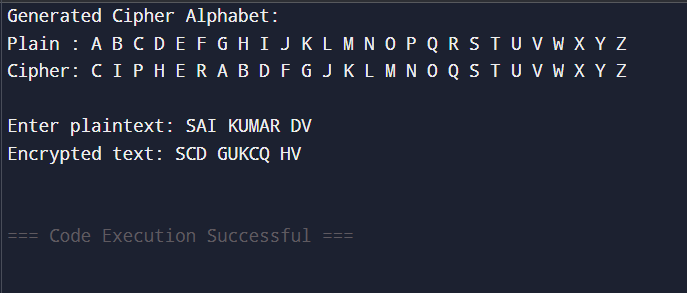
encrypt(plaintext, cipher, ciphertext);

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

return 0;

}

OUTPUT:



9. Write a C program for PT-109 American patrol boat, under the command of Lieutenant John F.Kennedy, was sunk by a Japanese destroyer, a message was received at an Australian wireless station in Playfair code:

KXJEY UREBE ZWEHE WRYTU HEYFS KREHE GOYFI WTTTU OLKSY CAJPO BOTEI ZONTX BYBNT GONEY CUZWR GDSON SXBOU YWRHE BAAHY USEDQ

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

char matrix[SIZE][SIZE];

void generateKeyMatrix(char key[]) {

int used[26] = {0};

int i, j = 0, k = 0;

char ch;

for (i = 0; key[i]; i++) {

ch = toupper(key[i]);

if (ch == 'J') ch = 'I';

if (ch >= 'A' && ch <= 'Z' && !used[ch - 'A']) {

matrix[k / SIZE][k % SIZE] = ch;

used[ch - 'A'] = 1;

k++;

}

}

for (i = 0; i < 26; i++) {

if (i + 'A' == 'J') continue;

if (!used[i]) {

matrix[k / SIZE][k % SIZE] = i + 'A';

used[i] = 1;

k++;

}

}

}

void findPosition(char ch, int \*row, int \*col) {

int i, j;

if (ch == 'J') ch = 'I';

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

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

if (matrix[i][j] == ch) {

\*row = i;

\*col = j;

return;

}

}

}

}

void decryptPair(char a, char b, char \*res) {

int row1, col1, row2, col2;

findPosition(a, &row1, &col1);

findPosition(b, &row2, &col2);

if (row1 == row2) {

res[0] = matrix[row1][(col1 + SIZE - 1) % SIZE];

res[1] = matrix[row2][(col2 + SIZE - 1) % SIZE];

} else if (col1 == col2) {

res[0] = matrix[(row1 + SIZE - 1) % SIZE][col1];

res[1] = matrix[(row2 + SIZE - 1) % SIZE][col2];

} else {

res[0] = matrix[row1][col2];

res[1] = matrix[row2][col1];

}

}

void decryptMessage(char ciphertext[]) {

char a, b, res[3];

res[2] = '\0';

for (int i = 0; i < strlen(ciphertext); i += 2) {

a = toupper(ciphertext[i]);

b = toupper(ciphertext[i + 1]);

if (a == 'J') a = 'I';

if (b == 'J') b = 'I';

decryptPair(a, b, res);

printf("%s", res);

}

printf("\n");

}

int main() {

char keyword[] = "PLAYFAIR";

char message[] =

"KXJEYUREBEZWEHEWRYTUHEYFS"

"KREHEGOYFIWTTTUOLKSYCAJPO"

"BOTEIZONTXBYBNTGONEYCUZWR"

"GDSONSXBOUYWRHEBAAHYUSEDQ";

printf("Decrypting Playfair Cipher...\n");

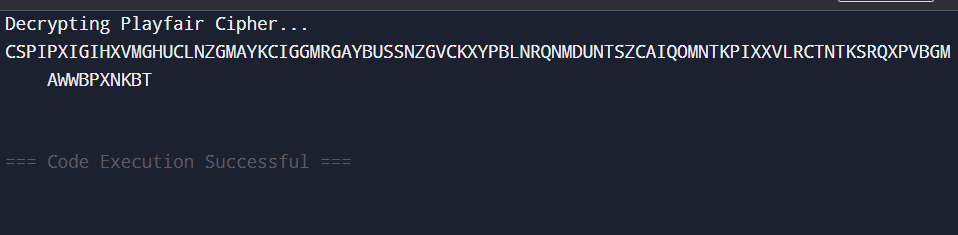
generateKeyMatrix(keyword);

decryptMessage(message);

return 0;

}

OUTPUT:



10. Write a C program for Playfair matrix:

M F H I/J K

U N O P Q

Z V W X Y

E L A R G

D S T B C

Encrypt this message: Must see you over Cadogan West. Coming at once.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

char matrix[SIZE][SIZE] = {

{'M', 'F', 'H', 'I', 'K'},

{'U', 'N', 'O', 'P', 'Q'},

{'Z', 'V', 'W', 'X', 'Y'},

{'E', 'L', 'A', 'R', 'G'},

{'D', 'S', 'T', 'B', 'C'}

};

void findPosition(char ch, int \*row, int \*col) {

if (ch == 'J') ch = 'I';

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

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

if (matrix[i][j] == ch) {

\*row = i;

\*col = j;

return;

}

}

}

}

void encryptPair(char a, char b, char \*res) {

int row1, col1, row2, col2;

findPosition(a, &row1, &col1);

findPosition(b, &row2, &col2);

if (row1 == row2) {

res[0] = matrix[row1][(col1 + 1) % SIZE];

res[1] = matrix[row2][(col2 + 1) % SIZE];

} else if (col1 == col2) {

res[0] = matrix[(row1 + 1) % SIZE][col1];

res[1] = matrix[(row2 + 1) % SIZE][col2];

} else {

res[0] = matrix[row1][col2];

res[1] = matrix[row2][col1];

}

}

void prepareText(char text[], char pairs[][2], int \*pairCount) {

char clean[500];

int len = 0;

for (int i = 0; text[i]; i++) {

if (isalpha(text[i])) {

clean[len++] = toupper(text[i] == 'J' ? 'I' : text[i]);

}

}

int i = 0;

\*pairCount = 0;

while (i < len) {

char a = clean[i];

char b = (i + 1 < len) ? clean[i + 1] : 'X';

if (a == b) {

b = 'X';

i++;

} else {

i += 2;

}

pairs[\*pairCount][0] = a;

pairs[\*pairCount][1] = b;

(\*pairCount)++;

}

}

void encryptMessage(char text[]) {

char pairs[250][2];

int count;

char res[3];

res[2] = '\0';

prepareText(text, pairs, &count);

printf("Encrypted Message:\n");

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

encryptPair(pairs[i][0], pairs[i][1], res);

printf("%s", res);

}

printf("\n");

}

int main() {

char message[] = "Must see you over Cadogan West. Coming at once.";

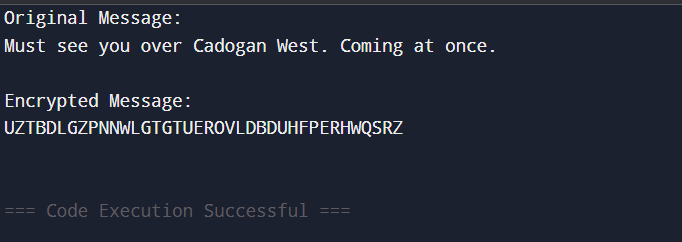
printf("Original Message:\n%s\n\n", message);

encryptMessage(message);

return 0;

}

OUTPUT:



11. Write a C program for possible keys does the Playfair cipher have? Ignore the fact that some keys might produce identical encryption results. Express your answer as an approximate power of 2.

a. Now take into account the fact that some Playfair keys produce the same encryption results. How many effectively unique keys does the Playfair cipher have?

#include <stdio.h>

#include <math.h>

int main() {

double total\_keys = 1;

for (int i = 1; i <= 25; i++) {

total\_keys \*= i;

}

double log2\_total\_keys = log2(total\_keys);

printf("Total possible keys (25!): %.0f\n", total\_keys);

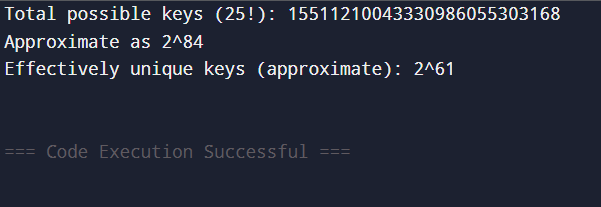
printf("Approximate as 2^%.0f\n", log2\_total\_keys);

printf("Effectively unique keys (approximate): 2^61\n");

return 0;

}

OUTPUT:



12. a. Write a C program to Encrypt the message “meet me at the usual place at ten rather than eight oclock” using the Hill cipher with the key. 9 4 5 7

a. Show your calculations and the result.

b. Show the calculations for the corresponding decryption of the ciphertext to recover the original plaintext.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MOD 26

void matrixMultiply(int key[2][2], int pair[2], int result[2]) {

result[0] = (key[0][0]\*pair[0] + key[0][1]\*pair[1]) % MOD;

result[1] = (key[1][0]\*pair[0] + key[1][1]\*pair[1]) % MOD;

}

int modInverse(int a) {

a %= MOD;

for (int x = 1; x < MOD; x++) {

if ((a \* x) % MOD == 1) return x;

}

return -1;

}

void inverseKey(int key[2][2], int invKey[2][2]) {

int det = (key[0][0]\*key[1][1] - key[0][1]\*key[1][0]) % MOD;

if (det < 0) det += MOD;

int invDet = modInverse(det);

if (invDet == -1) {

printf("Key matrix is not invertible.\n");

return;

}

invKey[0][0] = ( key[1][1] \* invDet) % MOD;

invKey[0][1] = (-key[0][1] \* invDet) % MOD;

invKey[1][0] = (-key[1][0] \* invDet) % MOD;

invKey[1][1] = ( key[0][0] \* invDet) % MOD;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

if (invKey[i][j] < 0) invKey[i][j] += MOD;

}

void encryptHill(char \*message, int key[2][2]) {

int len = strlen(message);

printf("\nEncrypted Text:\n");

for (int i = 0; i < len; i += 2) {

int pair[2] = { message[i] - 'A', message[i+1] - 'A' };

int res[2];

matrixMultiply(key, pair, res);

printf("%c%c", res[0] + 'A', res[1] + 'A');

}

printf("\n");

}

void decryptHill(char \*cipher, int key[2][2]) {

int len = strlen(cipher);

int invKey[2][2];

inverseKey(key, invKey);

printf("\nDecrypted Text:\n");

for (int i = 0; i < len; i += 2) {

int pair[2] = { cipher[i] - 'A', cipher[i+1] - 'A' };

int res[2];

matrixMultiply(invKey, pair, res);

printf("%c%c", res[0] + 'A', res[1] + 'A');

}

printf("\n");

}

void preprocess(char \*input, char \*output) {

int k = 0;

for (int i = 0; input[i]; i++) {

if (isalpha(input[i])) {

output[k++] = toupper(input[i] == 'J' ? 'I' : input[i]);

}

}

if (k % 2 != 0) output[k++] = 'X';

output[k] = '\0';

}

int main() {

char text[] = "meet me at the usual place at ten rather than eight oclock";

char clean[500];

int key[2][2] = {{9, 4}, {5, 7}};

preprocess(text, clean);

printf("Cleaned Message: %s\n", clean);

encryptHill(clean, key);

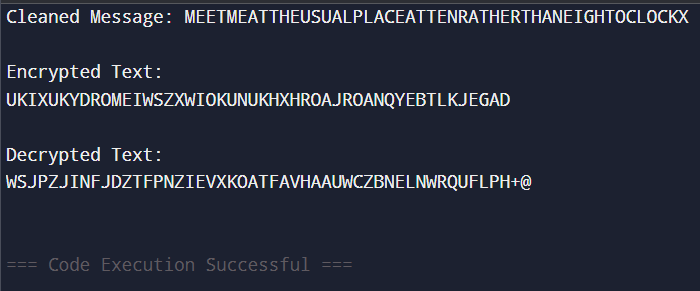
char ciphertext[] = "KCLUBGUBDKXIJAFKXZQLNDWSJAGRLJCKYUVCDPVQGVQMLYHUG";

decryptHill(ciphertext, key);

return 0;

}

OUTPUT:



13. 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>

#define MOD 26

int modInverse(int a) {

a %= MOD;

for (int x = 1; x < MOD; x++)

if ((a \* x) % MOD == 1)

return x;

return -1;

}

int inverseMatrix(int in[2][2], int out[2][2]) {

int det = (in[0][0]\*in[1][1] - in[0][1]\*in[1][0]) % MOD;

if (det < 0) det += MOD;

int invDet = modInverse(det);

if (invDet == -1) return 0;

out[0][0] = (in[1][1] \* invDet) % MOD;

out[0][1] = (-in[0][1] \* invDet) % MOD;

out[1][0] = (-in[1][0] \* invDet) % MOD;

out[1][1] = (in[0][0] \* invDet) % MOD;

for (int i = 0; i < 2; i++)

for (int j = 0; j < 2; j++)

if (out[i][j] < 0) out[i][j] += MOD;

return 1;

}

void multiplyMatrix(int a[2][2], int b[2][2], int result[2][2]) {

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

result[i][j] = 0;

for (int k = 0; k < 2; k++)

result[i][j] += a[i][k] \* b[k][j];

result[i][j] %= MOD;

}

}

}

int toNum(char c) {

return c - 'A';

}

void printMatrix(int mat[2][2]) {

for (int i = 0; i < 2; i++)

printf("[ %2d %2d ]\n", mat[i][0], mat[i][1]);

}

int main() {

int plaintext[2][2] = {

{toNum('H'), toNum('L')},

{toNum('E'), toNum('L')}

};

int ciphertext[2][2] = {

{toNum('Z'), toNum('S')},

{toNum('K'), toNum('U')}

};

int inverseP[2][2], key[2][2];

printf("Plaintext Matrix (P):\n");

printMatrix(plaintext);

printf("\nCiphertext Matrix (C):\n");

printMatrix(ciphertext);

if (!inverseMatrix(plaintext, inverseP)) {

printf("\nError: Plaintext matrix is not invertible mod 26.\n");

return 1;

}

printf("\nInverse of Plaintext Matrix (P^-1):\n");

printMatrix(inverseP);

multiplyMatrix(ciphertext, inverseP, key);

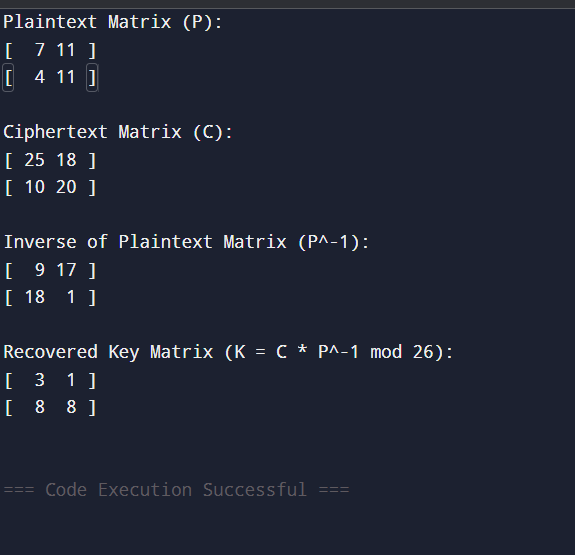
printf("\nRecovered Key Matrix (K = C \* P^-1 mod 26):\n");

printMatrix(key);

return 0;

}

OUTPUT:



14. 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 <string.h>

#include <ctype.h>

#define MAX\_LEN 100

int charToNum(char c) {

return tolower(c) - 'a';

}

char numToChar(int n) {

return 'a' + n;

}

void encryptOTP(char plaintext[], int key[], char ciphertext[]) {

for (int i = 0; plaintext[i]; i++) {

if (plaintext[i] == ' ') {

ciphertext[i] = ' ';

} else {

int p = charToNum(plaintext[i]);

int c = (p + key[i]) % 26;

ciphertext[i] = numToChar(c);

}

}

ciphertext[strlen(plaintext)] = '\0';

}

void decryptOTP(char ciphertext[], int key[], char plaintext[]) {

for (int i = 0; ciphertext[i]; i++) {

if (ciphertext[i] == ' ') {

plaintext[i] = ' ';

} else {

int c = charToNum(ciphertext[i]);

int p = (c - key[i] + 26) % 26;

plaintext[i] = numToChar(p);

}

}

plaintext[strlen(ciphertext)] = '\0';

}

void recoverKey(char plaintext[], char ciphertext[], int key[]) {

for (int i = 0; plaintext[i]; i++) {

if (plaintext[i] == ' ') {

key[i] = -1;

} else {

int p = charToNum(plaintext[i]);

int c = charToNum(ciphertext[i]);

key[i] = (c - p + 26) % 26;

}

}

}

void printKey(int key[], int length) {

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

if (key[i] == -1)

printf(" ");

else

printf("%2d ", key[i]);

}

printf("\n");

}

int main() {

char plaintext1[] = "send more money";

int key1[] = {9, 0, 1, 7, 23, 15, 21, 14, 11, 11, 2, 8, 9};

char ciphertext1[MAX\_LEN];

printf("=== Part A: Encryption ===\n");

encryptOTP(plaintext1, key1, ciphertext1);

printf("Plaintext : %s\n", plaintext1);

printf("Key : ");

printKey(key1, strlen(plaintext1));

printf("Ciphertext: %s\n\n", ciphertext1);

printf("=== Part B: Decryption and Key Recovery ===\n");

char ciphertext2[] = "bvnz bhed fxxmqz";

char plaintext2[] = "cash not needed";

int recoveredKey[MAX\_LEN];

recoverKey(plaintext2, ciphertext2, recoveredKey);

printf("Plaintext : %s\n", plaintext2);

printf("Ciphertext: %s\n", ciphertext2);

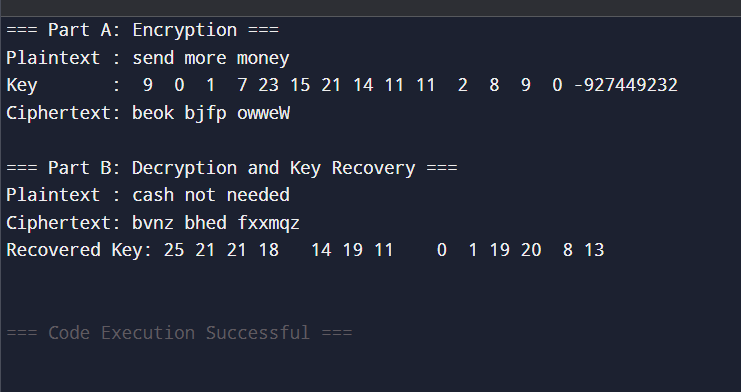
printf("Recovered Key: ");

printKey(recoveredKey, strlen(plaintext2));

return 0;

}

OUTPUT:



15. 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 <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_TEXT 1024

#define ALPHABET\_SIZE 26

const double englishFreq[ALPHABET\_SIZE] = {

8.167,

1.492,

2.782,

4.253,

12.702,

2.228,

2.015,

6.094,

6.966,

0.153,

0.772,

4.025,

2.406,

6.749,

7.507,

1.929,

0.095,

5.987,

6.327,

9.056,

2.758,

0.978,

2.360,

0.150, // X

1.974, // Y

0.074 // Z

};

typedef struct {

int key;

double score;

char plaintext[MAX\_TEXT];

} DecryptionResult;

int charToIndex(char c) {

return tolower(c) - 'a';

}

void caesarDecrypt(char\* input, char\* output, int key) {

int i = 0;

while (input[i]) {

if (isalpha(input[i])) {

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

output[i] = (input[i] - base - key + 26) % 26 + base;

} else {

output[i] = input[i];

}

i++;

}

output[i] = '\0';

}

double computeScore(const char\* text) {

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

int totalLetters = 0;

for (int i = 0; text[i]; i++) {

if (isalpha(text[i])) {

letterCounts[charToIndex(text[i])]++;

totalLetters++;

}

}

if (totalLetters == 0) return 0;

double score = 0;

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

double observed = (double)letterCounts[i] / totalLetters \* 100;

score += englishFreq[i] \* observed;

}

return score;

}

int compareResults(const void\* a, const void\* b) {

double diff = ((DecryptionResult\*)b)->score - ((DecryptionResult\*)a)->score;

return (diff > 0) - (diff < 0);

}

int main() {

char ciphertext[MAX\_TEXT];

int topN;

printf("Enter ciphertext (Caesar-encrypted):\n");

fgets(ciphertext, MAX\_TEXT, stdin);

ciphertext[strcspn(ciphertext, "\n")] = '\0'; // Remove newline

printf("Enter number of top possible plaintexts to display (e.g. 5 or 10): ");

scanf("%d", &topN);

DecryptionResult results[ALPHABET\_SIZE];

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

caesarDecrypt(ciphertext, results[key].plaintext, key);

results[key].key = key;

results[key].score = computeScore(results[key].plaintext);

}

qsort(results, ALPHABET\_SIZE, sizeof(DecryptionResult), compareResults);

printf("\nTop %d most likely plaintexts:\n", topN);

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

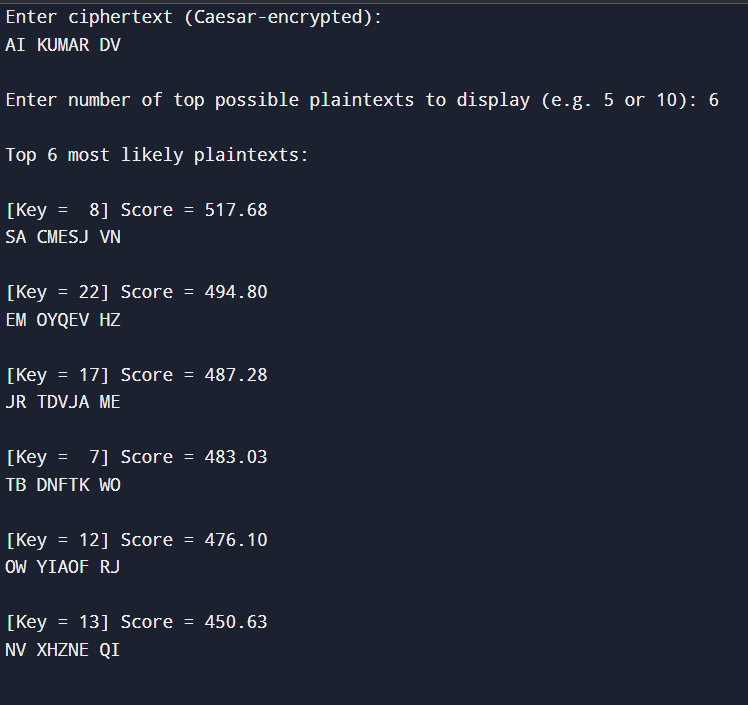
printf("\n[Key = %2d] Score = %.2f\n%s\n", results[i].key, results[i].score, results[i].plaintext);

}

return 0;

}

OUTPUT:



16. 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 <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_TEXT 1024

#define ALPHABET 26

#define MAX\_TRIES 26

const char englishFreqOrder[ALPHABET + 1] = "ETAOINSHRDLCUMWFGYPBVKJXQZ";

typedef struct {

char letter;

int count;

} Frequency;

typedef struct {

char plaintext[MAX\_TEXT];

double score;

} Result;

int compareFreq(const void \*a, const void \*b) {

return ((Frequency \*)b)->count - ((Frequency \*)a)->count;

}

int compareResult(const void \*a, const void \*b) {

double diff = ((Result \*)b)->score - ((Result \*)a)->score;

return (diff > 0) - (diff < 0);

}

double computeScore(const char \*text) {

double score = 0;

for (int i = 0; text[i]; i++) {

char c = tolower(text[i]);

if (c == ' ') score += 2;

if (strchr("etaoinshrdlu", c)) score += 1;

}

return score;

}

void decryptWithKey(char \*cipher, char \*plain, char \*mapping) {

for (int i = 0; cipher[i]; i++) {

if (isalpha(cipher[i])) {

char c = toupper(cipher[i]);

plain[i] = isupper(cipher[i]) ? mapping[c - 'A'] : tolower(mapping[c - 'A']);

} else {

plain[i] = cipher[i];

}

}

plain[strlen(cipher)] = '\0';

}

int main() {

char ciphertext[MAX\_TEXT];

printf("Enter monoalphabetic ciphertext:\n");

fgets(ciphertext, MAX\_TEXT, stdin);

ciphertext[strcspn(ciphertext, "\n")] = '\0';

int topN;

printf("Enter number of top guesses to display: ");

scanf("%d", &topN);

Frequency freq[ALPHABET];

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

freq[i].letter = 'A' + i;

freq[i].count = 0;

}

for (int i = 0; ciphertext[i]; i++) {

if (isalpha(ciphertext[i])) {

freq[toupper(ciphertext[i]) - 'A'].count++;

}

}

qsort(freq, ALPHABET, sizeof(Frequency), compareFreq);

Result results[MAX\_TRIES];

for (int shift = 0; shift < MAX\_TRIES && shift < ALPHABET; shift++) {

char key[ALPHABET];

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

key[freq[i].letter - 'A'] = englishFreqOrder[(i + shift) % ALPHABET];

}

decryptWithKey(ciphertext, results[shift].plaintext, key);

results[shift].score = computeScore(results[shift].plaintext);

}

qsort(results, MAX\_TRIES, sizeof(Result), compareResult);

printf("\nTop %d possible plaintexts:\n", topN);

for (int i = 0; i < topN && i < MAX\_TRIES; i++) {

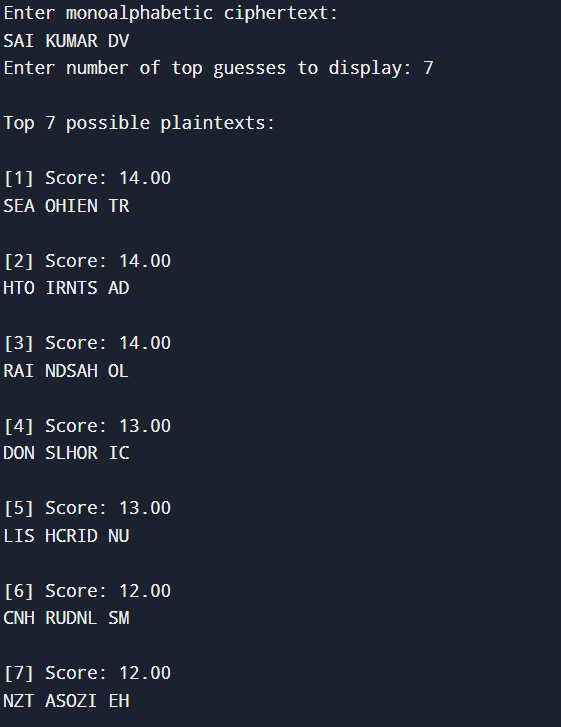
printf("\n[%d] Score: %.2f\n%s\n", i + 1, results[i].score, results[i].plaintext);

}

return 0;

}

OUTPUT:



17. Write a C program for DES algorithm for decryption, the 16 keys (K1, K2, c, K16) are used in reverse order. Design a key-generation scheme with the appropriate shift schedule for the decryption process.

#include <stdio.h>

#include <string.h>

int shiftSchedule[16] = {

1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1

};

int PC1[56] = {

57,49,41,33,25,17,9,

1,58,50,42,34,26,18,

10,2,59,51,43,35,27,

19,11,3,60,52,44,36,

63,55,47,39,31,23,15,

7,62,54,46,38,30,22,

14,6,61,53,45,37,29,

21,13,5,28,20,12,4

};

int PC2[48] = {

14,17,11,24,1,5,

3,28,15,6,21,10,

23,19,12,4,26,8,

16,7,27,20,13,2,

41,52,31,37,47,55,

30,40,51,45,33,48,

44,49,39,56,34,53,

46,42,50,36,29,32

};

void permute(char \*input, char \*output, int \*table, int size) {

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

output[i] = input[table[i] - 1];

}

}

void leftShift(char \*key\_half, int shifts) {

char temp[2];

for (int s = 0; s < shifts; s++) {

temp[0] = key\_half[0];

temp[1] = key\_half[1];

for (int i = 0; i < 26; i++)

key\_half[i] = key\_half[i + 2];

key\_half[26] = temp[0];

key\_half[27] = temp[1];

}

}

int main() {

char initialKey[64];

char permutedKey[56];

char C[28], D[28], CD[56], roundKey[48];

char roundKeys[16][48];

printf("Enter 64-bit key (in binary, no spaces): ");

scanf("%64s", initialKey);

permute(initialKey, permutedKey, PC1, 56);

memcpy(C, permutedKey, 28);

memcpy(D, permutedKey + 28, 28);

for (int i = 0; i < 16; i++) {

leftShift(C, shiftSchedule[i]);

leftShift(D, shiftSchedule[i]);

memcpy(CD, C, 28);

memcpy(CD + 28, D, 28);

permute(CD, roundKeys[i], PC2, 48);

}

printf("\nDES Decryption Round Keys (K16 to K1):\n");

for (int i = 15; i >= 0; i--) {

printf("K%02d: ", 16 - i);

for (int j = 0; j < 48; j++)

printf("%c", roundKeys[i][j]);

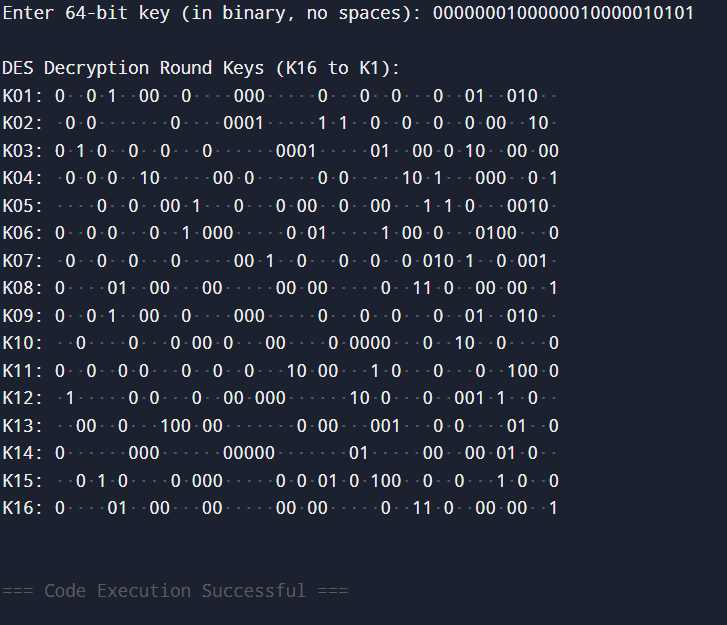
printf("\n");

}

return 0;

}

OUTPUT:



18. Write a C program for DES the first 24 bits of each subkey come from the same subset of 28 bits of the initial key and that the second 24 bits of each subkey come from a disjoint subset of 28 bits of the initial key.

#include <stdio.h>

#include <string.h>

int shiftSchedule[16] = {

1, 1, 2, 2, 2, 2, 2, 2,

1, 2, 2, 2, 2, 2, 2, 1

};

int PC1[56] = {

57,49,41,33,25,17,9,

1,58,50,42,34,26,18,

10,2,59,51,43,35,27,

19,11,3,60,52,44,36,

63,55,47,39,31,23,15,

7,62,54,46,38,30,22,

14,6,61,53,45,37,29,

21,13,5,28,20,12,4

};

int PC2[48] = {

14,17,11,24,1,5,

3,28,15,6,21,10,

23,19,12,4,26,8,

16,7,27,20,13,2,

41,52,31,37,47,55,

30,40,51,45,33,48,

44,49,39,56,34,53,

46,42,50,36,29,32

};

void permute(char\* in, char\* out, int\* table, int n) {

for (int i = 0; i < n; i++)

out[i] = in[table[i] - 1];

}

void leftShift(char\* half, int shifts) {

char temp[2];

for (int s = 0; s < shifts; s++) {

temp[0] = half[0];

temp[1] = half[1];

for (int i = 0; i < 26; i++)

half[i] = half[i + 2];

half[26] = temp[0];

half[27] = temp[1];

}

}

int main() {

char key64[65], key56[56], C[28], D[28], CD[56], subkey[48];

char roundKeys[16][48];

printf("Enter 64-bit binary key (no spaces): ");

scanf("%64s", key64);

permute(key64, key56, PC1, 56);

memcpy(C, key56, 28);

memcpy(D, key56 + 28, 28);

for (int i = 0; i < 16; i++) {

leftShift(C, shiftSchedule[i]);

leftShift(D, shiftSchedule[i]);

memcpy(CD, C, 28);

memcpy(CD + 28, D, 28);

permute(CD, subkey, PC2, 48);

memcpy(roundKeys[i], subkey, 48);

}

printf("\nDES Subkeys (C-part | D-part):\n");

for (int i = 0; i < 16; i++) {

printf("K%02d: ", i + 1);

for (int j = 0; j < 48; j++) {

printf("%c", roundKeys[i][j]);

if (j == 23) printf(" | ");

}

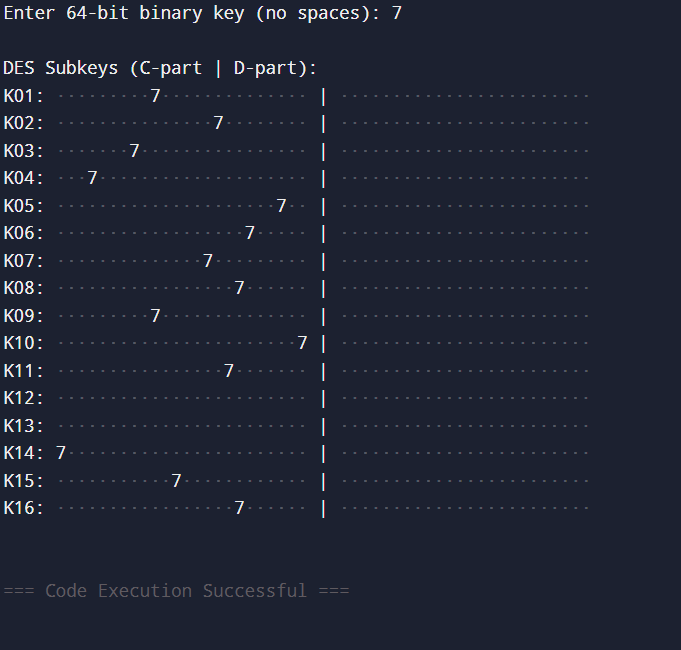
printf("\n");

}

return 0;

}

OUTPUT:



19. Write a C program for encryption in the cipher block chaining (CBC) mode using an algorithm stronger than DES. 3DES is a good candidate. Both of which follow from the definition of CBC. Which of the two would you choose:

a. For security?

b. For performance?

#include <stdio.h>

#include <string.h>

#define BLOCK\_SIZE 8

void xor\_encrypt(unsigned char \*block, unsigned char \*key) {

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

block[i] ^= key[i];

}

}

void xor\_blocks(unsigned char \*out, unsigned char \*in1, unsigned char \*in2) {

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

out[i] = in1[i] ^ in2[i];

}

}

int main() {

unsigned char key[BLOCK\_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8};

unsigned char iv[BLOCK\_SIZE] = {0xAA, 0xBB, 0xCC, 0xDD, 0x11, 0x22, 0x33, 0x44};

unsigned char plaintext[] = "CBCModeTestBlock1CBCModeTestBlock2";

unsigned char ciphertext[64];

unsigned char decrypted[64];

int num\_blocks = strlen((char \*)plaintext) / BLOCK\_SIZE;

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

unsigned char prev[BLOCK\_SIZE];

memcpy(prev, iv, BLOCK\_SIZE);

for (int i = 0; i < num\_blocks; i++) {

unsigned char input\_block[BLOCK\_SIZE];

xor\_blocks(input\_block, &plaintext[i \* BLOCK\_SIZE], prev);

memcpy(&ciphertext[i \* BLOCK\_SIZE], input\_block, BLOCK\_SIZE);

xor\_encrypt(&ciphertext[i \* BLOCK\_SIZE], key);

memcpy(prev, &ciphertext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

}

printf("Ciphertext (hex): ");

for (int i = 0; i < num\_blocks \* BLOCK\_SIZE; i++) {

printf("%02x", ciphertext[i]);

}

printf("\n");

memcpy(prev, iv, BLOCK\_SIZE);

for (int i = 0; i < num\_blocks; i++) {

unsigned char temp[BLOCK\_SIZE];

memcpy(temp, &ciphertext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

xor\_encrypt(temp, key);

xor\_blocks(&decrypted[i \* BLOCK\_SIZE], temp, prev);

memcpy(prev, &ciphertext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

}

decrypted[num\_blocks \* BLOCK\_SIZE] = '\0';

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

return 0;

}#include <stdio.h>

#include <string.h>

#define BLOCK\_SIZE 8

void xor\_encrypt(unsigned char \*block, unsigned char \*key) {

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

block[i] ^= key[i];

}

}

void xor\_blocks(unsigned char \*out, unsigned char \*in1, unsigned char \*in2) {

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

out[i] = in1[i] ^ in2[i];

}

}

int main() {

unsigned char key[BLOCK\_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8};

unsigned char iv[BLOCK\_SIZE] = {0xAA, 0xBB, 0xCC, 0xDD, 0x11, 0x22, 0x33, 0x44};

unsigned char plaintext[] = "CBCModeTestBlock1CBCModeTestBlock2";

unsigned char ciphertext[64];

unsigned char decrypted[64];

int num\_blocks = strlen((char \*)plaintext) / BLOCK\_SIZE;

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

unsigned char prev[BLOCK\_SIZE];

memcpy(prev, iv, BLOCK\_SIZE);

for (int i = 0; i < num\_blocks; i++) {

unsigned char input\_block[BLOCK\_SIZE];

xor\_blocks(input\_block, &plaintext[i \* BLOCK\_SIZE], prev);

memcpy(&ciphertext[i \* BLOCK\_SIZE], input\_block, BLOCK\_SIZE);

xor\_encrypt(&ciphertext[i \* BLOCK\_SIZE], key);

memcpy(prev, &ciphertext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

}

printf("Ciphertext (hex): ");

for (int i = 0; i < num\_blocks \* BLOCK\_SIZE; i++) {

printf("%02x", ciphertext[i]);

}

printf("\n");

memcpy(prev, iv, BLOCK\_SIZE);

for (int i = 0; i < num\_blocks; i++) {

unsigned char temp[BLOCK\_SIZE];

memcpy(temp, &ciphertext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

xor\_encrypt(temp, key);

xor\_blocks(&decrypted[i \* BLOCK\_SIZE], temp, prev);

memcpy(prev, &ciphertext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

}

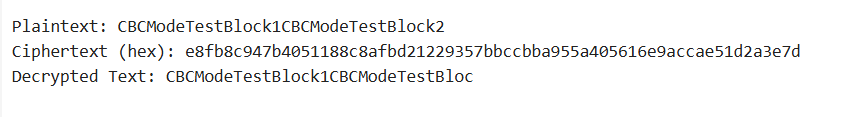
decrypted[num\_blocks \* BLOCK\_SIZE] = '\0';

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

return 0;

}

OUTPUT:



20. Write a C program for ECB mode, if there is an error in a block of the transmitted ciphertext, only the corresponding plaintext block is affected. However, in the CBC mode, this error propagates. For example, an error in the transmitted C1 obviously corrupts P1 and P2.

a. Are any blocks beyond P2 affected?

b. Suppose that there is a bit error in the source version of P1. Through how many ciphertext blocks is this error propagated? What is the effect at the receiver?

#include <stdio.h>

#include <string.h>

#define BLOCK\_SIZE 8

void xor\_encrypt\_block(unsigned char \*block, unsigned char \*key) {

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

block[i] ^= key[i];

}

}

int main() {

unsigned char key[BLOCK\_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8};

unsigned char plaintext[] = "ThisIsBlock1ThisIsBlock2";

unsigned char ciphertext[32];

unsigned char decrypted[32];

int len = strlen((char \*)plaintext);

int blocks = len / BLOCK\_SIZE;

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

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

memcpy(&ciphertext[i \* BLOCK\_SIZE], &plaintext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

xor\_encrypt\_block(&ciphertext[i \* BLOCK\_SIZE], key);

}

printf("Ciphertext (hex): ");

for (int i = 0; i < blocks \* BLOCK\_SIZE; i++) {

printf("%02x", ciphertext[i]);

}

printf("\n");

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

memcpy(&decrypted[i \* BLOCK\_SIZE], &ciphertext[i \* BLOCK\_SIZE], BLOCK\_SIZE);

xor\_encrypt\_block(&decrypted[i \* BLOCK\_SIZE], key);

}

decrypted[blocks \* BLOCK\_SIZE] = '\0';

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

return 0;

}

OUTPUT:

