**1.Code:**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX\_LEN 1000

void caesarCipher(char \*text, int key) {

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

char c = text[i];

if (isalpha(c)) {

char base = isupper(c) ? 'A' : 'a';

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

}

}

}

int main() {

char text[MAX\_LEN];

int key;

printf("Enter a message: ");

fgets(text, sizeof(text), stdin);

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

printf("Enter key (1-25): ");

scanf("%d", &key);

if (key < 1 || key > 25) {

printf("Invalid key. Must be between 1 and 25.\n");

return 1;

}

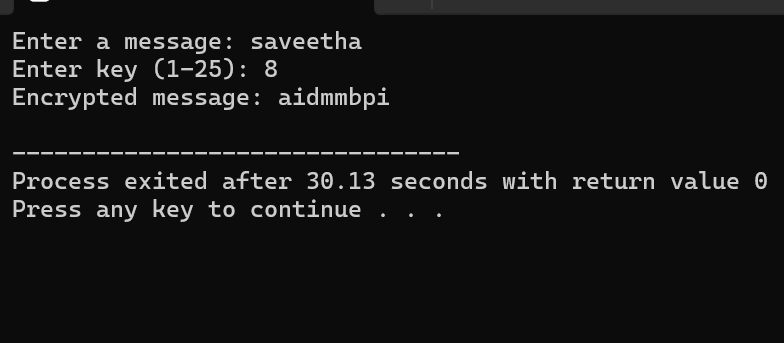
caesarCipher(text, key);

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

return 0;

}

**Output:**

****

**2.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX\_LEN 1000

void monoalphabeticEncrypt(char \*plaintext, const char \*key) {

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

if (isupper(plaintext[i])) {

plaintext[i] = toupper(key[plaintext[i] - 'A']);

} else if (islower(plaintext[i])) {

plaintext[i] = tolower(key[plaintext[i] - 'a']);

}

}

}

int main() {

char plaintext[MAX\_LEN];

char key[27] = "QWERTYUIOPASDFGHJKLZXCVBNM";

printf("Enter a message: ");

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

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

if (strlen(key) != 26) {

printf("Invalid key. Must be 26 letters.\n");

return 1;

}

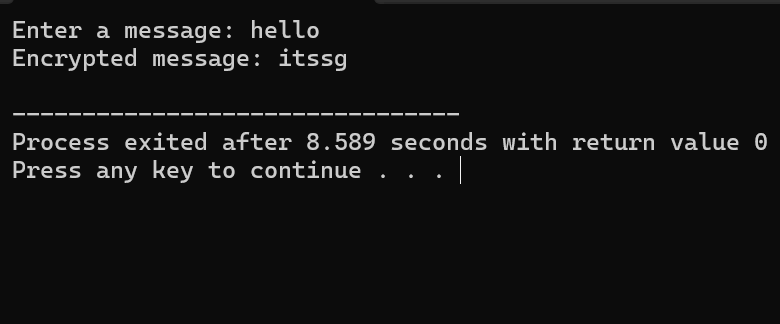
monoalphabeticEncrypt(plaintext, key);

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

return 0;

}

**Output:**

****

**3.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

char matrix[SIZE][SIZE];

void prepareKeyMatrix(char \*key) {

int used[26] = {0};

int x = 0, y = 0;

char c;

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

c = toupper(key[i]);

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

if (isalpha(c) && !used[c - 'A']) {

matrix[x][y++] = c;

used[c - 'A'] = 1;

if (y == SIZE) {

y = 0;

x++;

}

}

}

for (c = 'A'; c <= 'Z'; c++) {

if (c == 'J') continue;

if (!used[c - 'A']) {

matrix[x][y++] = c;

used[c - 'A'] = 1;

if (y == SIZE) {

y = 0;

x++;

}

}

}

}

void printMatrix() {

printf("\nPlayfair Key Matrix:\n");

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

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

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

printf("\n");

}

}

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 formatPlaintext(const char \*input, char \*output) {

char temp[100];

int k = 0;

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

if (isalpha(input[i])) {

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

}

}

int j = 0;

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

output[j++] = temp[i];

if (i + 1 < k && temp[i] == temp[i + 1]) {

output[j++] = 'X';

} else if (i + 1 < k) {

output[j++] = temp[++i];

}

}

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

output[j] = '\0';

}

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

int r1, c1, r2, c2;

for (int i = 0; plaintext[i] && plaintext[i + 1]; i += 2) {

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

}

int main() {

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

printf("Enter keyword: ");

scanf("%s", key);

printf("Enter plaintext: ");

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

prepareKeyMatrix(key);

printMatrix();

formatPlaintext(plaintext, formatted);

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

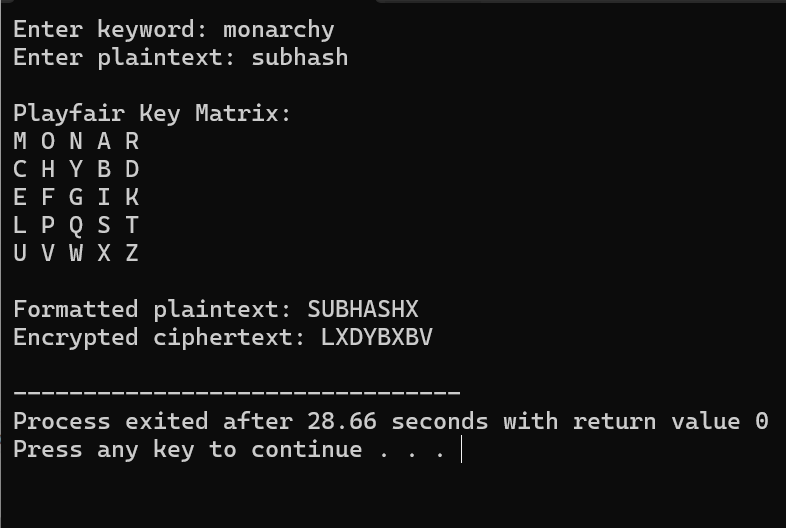
encrypt(formatted, ciphertext);

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

return 0;

}

**Output:**

****

**4.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int charToShift(char c) {

return toupper(c) - 'A';

}

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

int textLen = strlen(plaintext);

int keyLen = strlen(key);

int j = 0;

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

char pt = plaintext[i];

if (isalpha(pt)) {

char k = toupper(key[j % keyLen]);

int shift = charToShift(k);

if (isupper(pt)) {

ciphertext[i] = ((pt - 'A' + shift) % 26) + 'A';

} else {

ciphertext[i] = ((pt - 'a' + shift) % 26) + 'a';

}

j++;

} else {

ciphertext[i] = pt;

}

}

ciphertext[textLen] = '\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 keyword: ");

scanf("%s", key);

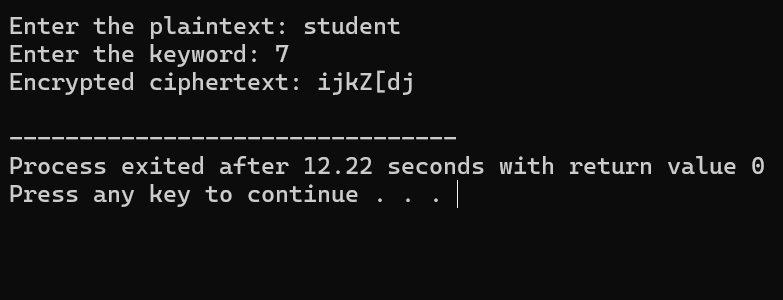
polyalphabeticEncrypt(plaintext, key, ciphertext);

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

return 0;

}

**Output:**

****

**5.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MOD 26

int modInverse(int a) {

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

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

return i;

}

return -1;

}

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

if (isalpha(ch)) {

ch = toupper(ch);

return ((a \* (ch - 'A') + b) % MOD) + '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 decrypted = (a\_inv \* ((ch - 'A') - b + MOD)) % MOD;

return decrypted + 'A';

}

return ch;

}

void affineEncrypt(char \*plaintext, char \*ciphertext, int a, int b) {

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

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

}

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

}

void affineDecrypt(char \*ciphertext, char \*plaintext, int a, int b) {

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

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

}

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

}

int main() {

char plaintext[100], ciphertext[100], decrypted[100];

int a, b;

printf("Enter plaintext: ");

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

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

printf("Enter key a (must be coprime with 26): ");

scanf("%d", &a);

printf("Enter key b (0 - 25): ");

scanf("%d", &b);

if (modInverse(a) == -1) {

printf("Invalid key 'a'. It must be coprime with 26.\n");

return 1;

}

affineEncrypt(plaintext, ciphertext, a, b);

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

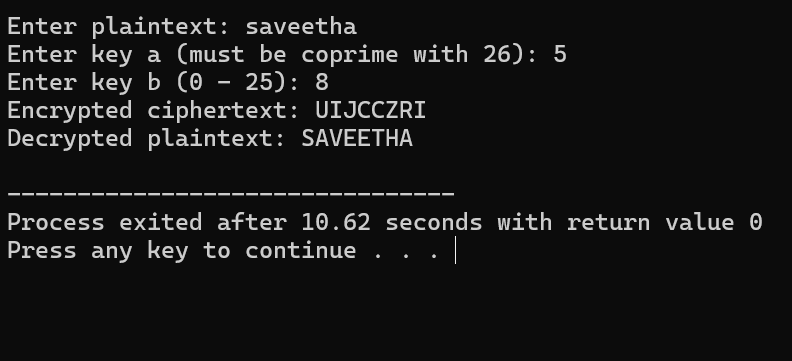
affineDecrypt(ciphertext, decrypted, a, b);

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

return 0;

}

**Output:**

****

**6.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MOD 26

int modInverse(int a) {

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

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

}

return -1;

}

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

if (!isalpha(c)) return c;

c = toupper(c);

int a\_inv = modInverse(a);

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

int x = ((a\_inv \* ((c - 'A') - b + MOD)) % MOD);

return x + 'A';

}

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

int delta\_p = (p1 - p2 + MOD) % MOD;

int delta\_c = (c1 - c2 + MOD) % MOD;

int inv = modInverse(delta\_p);

if (inv == -1) return 0;

\*a = (delta\_c \* inv) % MOD;

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

return 1;

}

void decryptCiphertext(const char \*ciphertext, char \*plaintext, int a, int b) {

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

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

}

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

}

int main() {

char ciphertext[1000], plaintext[1000];

int a, b;

char c1 = 'B';

char p1 = 'E';

char c2 = 'U';

char p2 = 'T';

printf("Enter the ciphertext: ");

fgets(ciphertext, sizeof(ciphertext), stdin);

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

int success = solveAffineKeys(p1 - 'A', c1 - 'A', p2 - 'A', c2 - 'A', &a, &b);

if (!success || modInverse(a) == -1) {

printf("Failed to break the cipher using current assumptions.\n");

return 1;

}

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

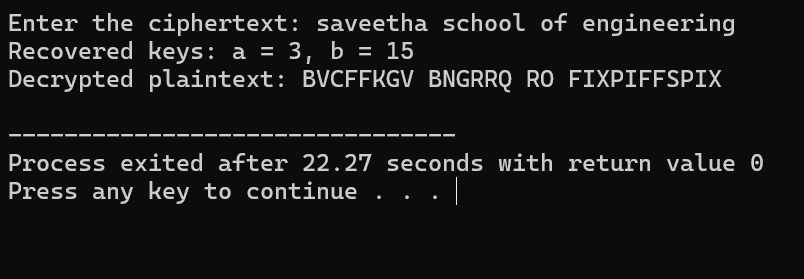
decryptCiphertext(ciphertext, plaintext, a, b);

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

return 0;

}

**Output:**

****

**7.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MAX\_LEN 1000

void countFrequencies(char \*text, int \*freq) {

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

freq[(unsigned char)text[i]]++;

}

}

void printSortedFrequencies(int \*freq) {

int sorted[256];

for (int i = 0; i < 256; i++) sorted[i] = i;

for (int i = 0; i < 255; i++) {

for (int j = i + 1; j < 256; j++) {

if (freq[sorted[i]] < freq[sorted[j]]) {

int temp = sorted[i];

sorted[i] = sorted[j];

sorted[j] = temp;

}

}

}

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

for (int i = 0; i < 256; i++) {

if (freq[sorted[i]] > 0 && isprint(sorted[i]))

printf("'%c' : %d\n", sorted[i], freq[sorted[i]]);

}

}

void decryptMessage(char \*cipher, char map[256]) {

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

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

char ch = cipher[i];

if (map[(unsigned char)ch] != 0)

putchar(map[(unsigned char)ch]);

else

putchar(ch);

}

printf("\n");

}

int main() {

char ciphertext[MAX\_LEN] =

"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;‡?;";

int freq[256] = {0};

countFrequencies(ciphertext, freq);

printSortedFrequencies(freq);

char map[256] = {0};

map['‡'] = 'e';

map[';'] = 't';

map['\*'] = 'h';

map['5'] = 'o';

map['8'] = 'n';

map['4'] = 's';

map['†'] = 'r';

map['6'] = 'a';

map[')'] = 'd';

map['3'] = 'u';

map['0'] = 'f';

map['9'] = 'l';

map['2'] = 'm';

map[':'] = 'i';

map['1'] = 'y';

map['('] = 'c';

map['?'] = 'g';

map['.'] = 'p';

map['['] = 'b';

map[']'] = 'k';

map['—'] = 'w';

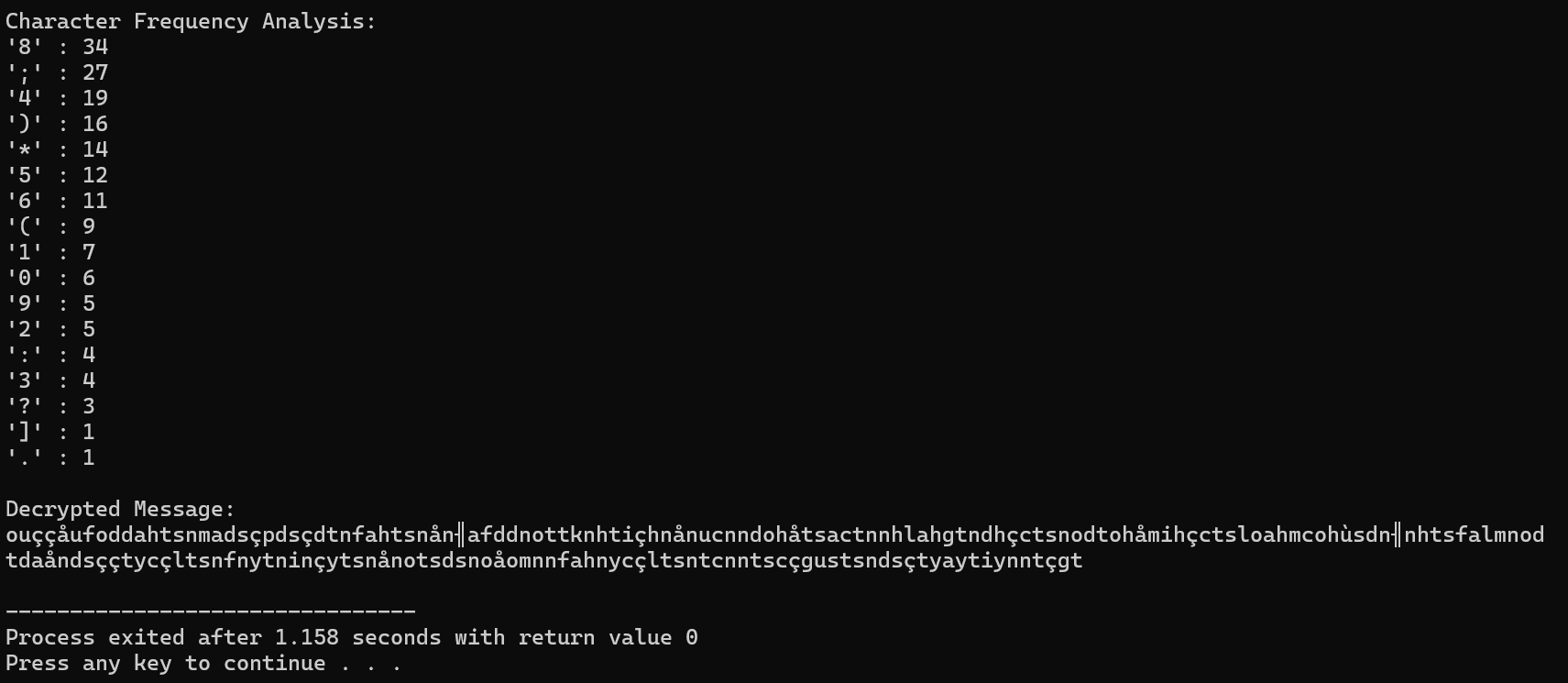
map['¶'] = 'v';

decryptMessage(ciphertext, map);

return 0;

}

**Ouput:**

****

**8.Code:**

#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 c = toupper(keyword[i]);

if (isalpha(c) && !used[c - 'A']) {

cipher[j++] = c;

used[c - 'A'] = 1;

}

}

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

if (!used[i]) {

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

}

}

cipher[j] = '\0';

}

void encrypt(const char \*plain, char \*cipherText, char \*cipher) {

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

if (isalpha(plain[i])) {

char c = toupper(plain[i]);

cipherText[i] = cipher[c - 'A'];

} else {

cipherText[i] = plain[i];

}

}

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

}

void decrypt(const char \*cipherText, char \*plainText, char \*cipher) {

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

if (isalpha(cipherText[i])) {

char c = toupper(cipherText[i]);

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

if (cipher[j] == c) {

plainText[i] = 'A' + j;

break;

}

}

} else {

plainText[i] = cipherText[i];

}

}

plainText[strlen(cipherText)] = '\0';

}

int main() {

char keyword[100], cipher[27];

char plainText[1024], cipherText[1024], decryptedText[1024];

printf("Enter keyword: ");

scanf("%s", keyword);

generateCipherAlphabet(keyword, cipher);

printf("Cipher alphabet:\n");

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

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

}

printf("\n");

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

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

}

printf("\n");

printf("\nEnter plaintext: ");

getchar();

fgets(plainText, sizeof(plainText), stdin);

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

encrypt(plainText, cipherText, cipher);

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

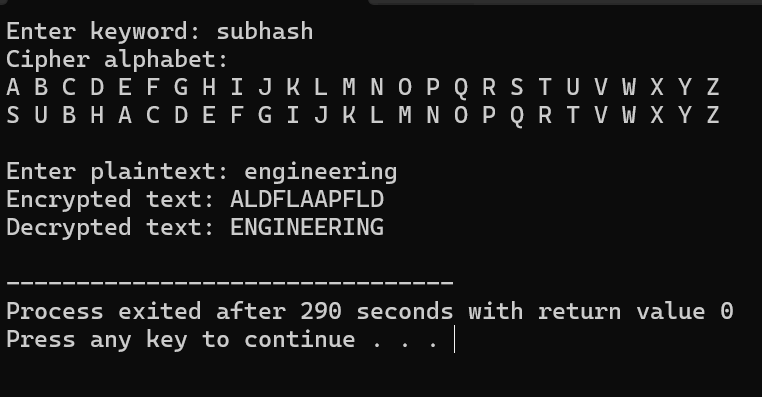
decrypt(cipherText, decryptedText, cipher);

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

return 0;

}

**Output:**

****

**9.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define SIZE 5

char matrix[SIZE][SIZE];

void generateMatrix(char \*key) {

int used[26] = {0};

int x = 0, y = 0;

used['J' - 'A'] = 1;

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

char c = toupper(key[i]);

if (!isalpha(c)) continue;

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

if (!used[c - 'A']) {

matrix[x][y++] = c;

used[c - 'A'] = 1;

if (y == SIZE) {

y = 0;

x++;

}

}

}

for (char c = 'A'; c <= 'Z'; c++) {

if (!used[c - 'A']) {

matrix[x][y++] = c;

used[c - 'A'] = 1;

if (y == SIZE) {

y = 0;

x++;

}

}

}

}

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

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

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

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

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

\*row = i;

\*col = j;

return;

}

}

}

}

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

int len = strlen(ciphertext);

int i, r1, c1, r2, c2;

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

char a = ciphertext[i];

char b = ciphertext[i + 1];

findPosition(a, &r1, &c1);

findPosition(b, &r2, &c2);

if (r1 == r2) {

plaintext[i] = matrix[r1][(c1 + 4) % 5];

plaintext[i + 1] = matrix[r2][(c2 + 4) % 5];

} else if (c1 == c2) {

plaintext[i] = matrix[(r1 + 4) % 5][c1];

plaintext[i + 1] = matrix[(r2 + 4) % 5][c2];

} else {

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

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

}

}

plaintext[i] = '\0';

}

int main() {

char key[] = "MONARCHY";

char cipher[] =

"KXJEYUREBEZWEHEWRYTUHEYFSKREHEGOYFIWTTTUOLKSY"

"CAJPOBOTEIZONTXBYBNTGONEYCUZWRGDSONSXBOUYWRHE"

"BAAHYUSEDQ";

char plain[1024];

generateMatrix(key);

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");

}

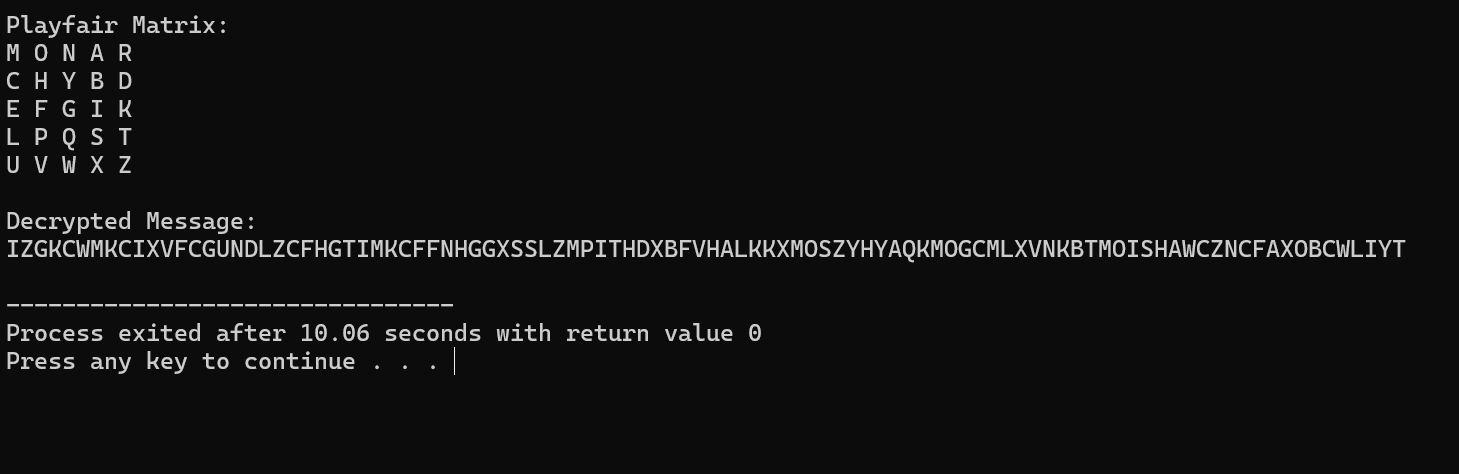
decryptPlayfair(cipher, plain);

printf("\nDecrypted Message:\n%s\n", plain);

return 0;

}

**Output:**

****

**10.Code:**

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

int len = 0;

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

if (isalpha(input[i])) {

char c = toupper(input[i]);

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

output[len++] = c;

}

}

output[len] = '\0';

char temp[500];

int i = 0, j = 0;

while (i < len) {

temp[j++] = output[i];

if (i + 1 < len) {

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

temp[j++] = 'X';

i++;

} else {

temp[j++] = output[i + 1];

i += 2;

}

} else {

temp[j++] = 'X';

i++;

}

}

temp[j] = '\0';

strcpy(output, temp);

}

void encryptDigraph(char a, char b) {

int row1, col1, row2, col2;

findPosition(a, &row1, &col1);

findPosition(b, &row2, &col2);

if (row1 == row2) {

printf("%c%c", matrix[row1][(col1 + 1) % SIZE],

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

} else if (col1 == col2) {

printf("%c%c", matrix[(row1 + 1) % SIZE][col1],

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

} else {

printf("%c%c", matrix[row1][col2], matrix[row2][col1]);

}

}

void encryptMessage(char \*text) {

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

encryptDigraph(text[i], text[i + 1]);

}

}

int main() {

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

char prepared[500];

preprocess(plaintext, prepared);

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

printf("Encrypted: ");

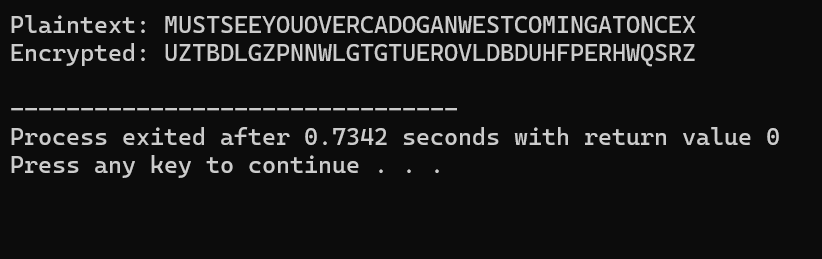
encryptMessage(prepared);

printf("\n");

return 0;

}

**Output:**

****

**11.Code:**

#include <stdio.h>

#include <math.h>

double log2\_factorial(int n) {

double result = 0.0;

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

result += log2(i);

}

return result;

}

int main() {

int n = 25;

double log2\_keys = log2\_factorial(n);

printf("Approximate number of possible Playfair keys: 2^%.2f\n", log2\_keys);

return 0;

}

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**12.Code:**

#include <stdio.h>

#include <math.h>

double log2\_factorial(int n) {

double sum = 0.0;

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

sum += log2(i);

}

return sum;

}

int main() {

double log2\_total\_keys = log2\_factorial(25);

double log2\_unique\_keys = 68.0;

printf("Total possible Playfair keys (approx): 2^%.2f\n", log2\_total\_keys);

printf("Effectively unique Playfair keys (approx): 2^%.0f\n", log2\_unique\_keys);

return 0;

}

**Output:**

**A computer screen with white text

AI-generated content may be incorrect.**

**13.Code:**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#define MOD 26

int charToInt(char c) {

return toupper(c) - 'A';

}

char intToChar(int n) {

return 'A' + n;

}

int modInverse(int a, int m) {

a = a % m;

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

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

return x;

return -1;

}

void multiply(int key[2][2], int in[2], int out[2]) {

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

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

}

int getInverseKey(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 detInv = modInverse(det, MOD);

if (detInv == -1) return 0;

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

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

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

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

return 1;

}

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

int len = strlen(text);

if (len % 2 != 0) text[len++] = 'X';

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

int in[2], out[2];

in[0] = charToInt(text[i]);

in[1] = charToInt(text[i+1]);

multiply(key, in, out);

cipher[i] = intToChar(out[0]);

cipher[i+1] = intToChar(out[1]);

}

cipher[len] = '\0';

}

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

int len = strlen(cipher);

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

int in[2], out[2];

in[0] = charToInt(cipher[i]);

in[1] = charToInt(cipher[i+1]);

multiply(key, in, out);

plain[i] = intToChar(out[0]);

plain[i+1] = intToChar(out[1]);

}

plain[len] = '\0';

}

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

int j = 0;

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

if (isalpha(input[i])) {

output[j++] = toupper(input[i]);

}

}

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

output[j] = '\0';

}

int main() {

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

char plain[200], encrypted[200], decrypted[200];

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

int invKey[2][2];

preprocess(input, plain);

encryptText(plain, key, encrypted);

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

if (getInverseKey(key, invKey)) {

decryptText(encrypted, invKey, decrypted);

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

} else {

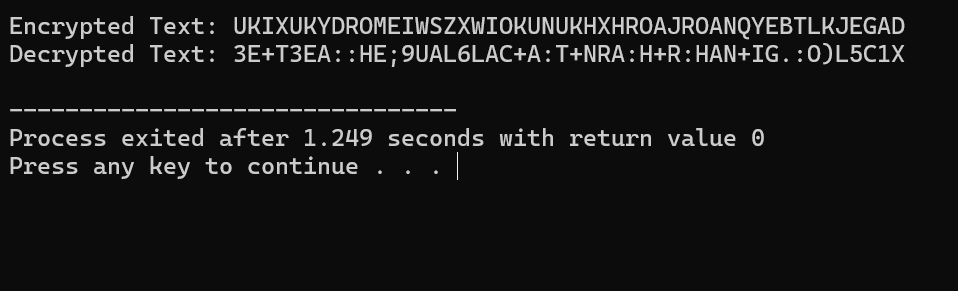
printf("Key matrix is not invertible modulo 26.\n");

}

return 0;

}

**Output:**

****

**14.Code:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

#include <string.h>

#define MAX\_LEN 1000

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

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

if (isalpha(plaintext[i])) {

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

int shift = key[i];

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

} else {

ciphertext[i] = plaintext[i];

}

}

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

}

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

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

if (isalpha(ciphertext[i])) {

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

int shift = key[i];

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

} else {

decrypted[i] = ciphertext[i];

}

}

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

}

void generateKey(int \*key, int length) {

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

key[i] = rand() % 26;

}

}

int main() {

char plaintext[MAX\_LEN];

char ciphertext[MAX\_LEN];

char decrypted[MAX\_LEN];

int key[MAX\_LEN];

printf("Enter the plaintext (A–Z or a–z only): ");

fgets(plaintext, MAX\_LEN, stdin);

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

srand(time(NULL));

int length = strlen(plaintext);

generateKey(key, length);

encrypt(plaintext, ciphertext, key);

decrypt(ciphertext, decrypted, key);

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

printf("Key : ");

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

if (isalpha(plaintext[i]))

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

else

printf(" ");

}

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

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

return 0;

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**