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.

Program:

#include<stdio.h>

#include<ctype.h>

int main() {

char text[500], ch;

int key;

printf("Enter a message to encrypt: ");

scanf("%s", text);

printf("Enter the key: ");

scanf("%d", & key);

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

ch = text[i];

if (isalnum(ch)) {

if (islower(ch)) {

ch = (ch - 'a' + key) % 26 + 'a';

}

if (isupper(ch)) {

ch = (ch - 'A' + key) % 26 + 'A';

}

if (isdigit(ch)) {

ch = (ch - '0' + key) % 10 + '0';

}

}

else {

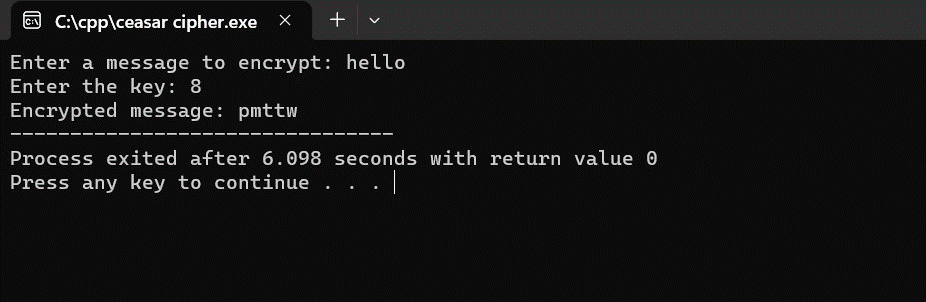
printf("Invalid Message");

}

text[i] = ch; }

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

return 0;}



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.

Program:

#include<stdio.h>

char monocipher\_encr(char);

char alpha[27][3] = { { 'a', 'f' }, { 'b', 'a' }, { 'c', 'g' }, { 'd', 'u' }, {

'e', 'n' }, { 'f', 'i' }, { 'g', 'j' }, { 'h', 'k' }, { 'i', 'l' }, {

'j', 'm' }, { 'k', 'o' }, { 'l', 'p' }, { 'm', 'q' }, { 'n', 'r' }, {

'o', 's' }, { 'p', 't' }, { 'q', 'v' }, { 'r', 'w' }, { 's', 'x' }, {

't', 'y' }, { 'u', 'z' }, { 'v', 'b' }, { 'w', 'c' }, { 'x', 'd' }, {

'y', 'e' }, { 'z', 'h' } };

char str[20];

int main() {

char str[20], str2[20];

int i;

printf("\n Enter String..");

gets(str);

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

str2[i] = monocipher\_encr(str[i]);

}

str2[i] = '\0';

printf("\n Before ecryption..%s", str);

printf("\n After ecryption..%s\n", str2);

}

char monocipher\_encr(char a) {

int i;

for (i = 0; i < 27; i++) {

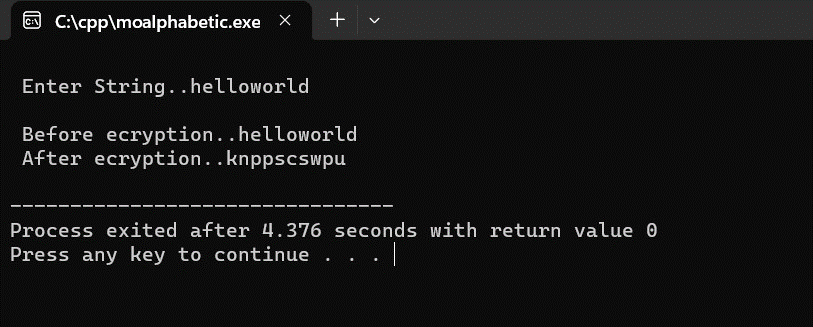
if (a == alpha[i][0])

break;

}

return alpha[i][1];

}



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.

Program:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 30

void toLowerCase(char plain[], int ps)

{

int i;

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

if (plain[i] > 64 && plain[i] < 91)

plain[i] += 32;

}

}

int removeSpaces(char\* plain, int ps)

{

int i, count = 0;

for (i = 0; i < ps; i++)

if (plain[i] != ' ')

plain[count++] = plain[i];

plain[count] = '\0';

return count;

}

void generateKeyTable(char key[], int ks, char keyT[5][5])

{

int i, j, k, flag = 0, \*dicty;

dicty = (int\*)calloc(26, sizeof(int));

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

if (key[i] != 'j')

dicty[key[i] - 97] = 2;

}

dicty['j' - 97] = 1;

i = 0;

j = 0;

for (k = 0; k < ks; k++) {

if (dicty[key[k] - 97] == 2) {

dicty[key[k] - 97] -= 1;

keyT[i][j] = key[k];

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

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

if (dicty[k] == 0) {

keyT[i][j] = (char)(k + 97);

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

}

void search(char keyT[5][5], char a, char b, int arr[])

{

int i, j;

if (a == 'j')

a = 'i';

else if (b == 'j')

b = 'i';

for (i = 0; i < 5; i++) {

for (j = 0; j < 5; j++) {

if (keyT[i][j] == a) {

arr[0] = i;

arr[1] = j;

}

else if (keyT[i][j] == b) {

arr[2] = i;

arr[3] = j;

}

}

}

}

int mod5(int a) { return (a % 5); }

int prepare(char str[], int ptrs)

{

if (ptrs % 2 != 0) {

str[ptrs++] = 'z';

str[ptrs] = '\0';

}

return ptrs;

}

void encrypt(char str[], char keyT[5][5], int ps)

{

int i, a[4];

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

search(keyT, str[i], str[i + 1], a);

if (a[0] == a[2]) {

str[i] = keyT[a[0]][mod5(a[1] + 1)];

str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];

}

else if (a[1] == a[3]) {

str[i] = keyT[mod5(a[0] + 1)][a[1]];

str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];

}

else {

str[i] = keyT[a[0]][a[3]];

str[i + 1] = keyT[a[2]][a[1]];

}

}

}

void encryptByPlayfairCipher(char str[], char key[])

{

char ps, ks, keyT[5][5];

ks = strlen(key);

ks = removeSpaces(key, ks);

toLowerCase(key, ks);

ps = strlen(str);

toLowerCase(str, ps);

ps = removeSpaces(str, ps);

ps = prepare(str, ps);

generateKeyTable(key, ks, keyT);

encrypt(str, keyT, ps);

}

int main()

{

char str[SIZE], key[SIZE];

strcpy(key, "Monarchy");

printf("Key text: %s\n", key);

strcpy(str, "instruments");

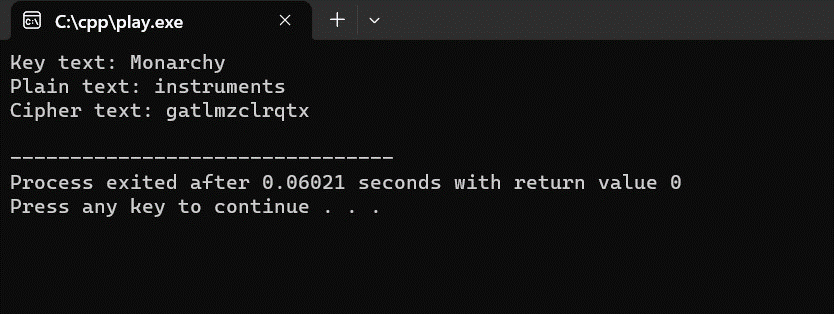
printf("Plain text: %s\n", str);

encryptByPlayfairCipher(str, key);

printf("Cipher text: %s\n", str);

return 0;

}



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

Program:

#include <stdio.h>

#include <string.h>

char encryptChar(char ch, char key) {

if (isalpha(ch)) {

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

char encrypted = (ch - base + (key - 'A')) % 26 + base;

return encrypted;

}

return ch;

}

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

int textLength = strlen(plaintext);

int keyLength = strlen(key);

char encryptedText[textLength + 1];

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

char keyChar = key[i % keyLength];

encryptedText[i] = encryptChar(plaintext[i], keyChar);

}

encryptedText[textLength] = '\0';

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

}

int main() {

char plaintext[1000];

char key[100];

printf("Enter the plaintext: ");

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

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

printf("Enter the key: ");

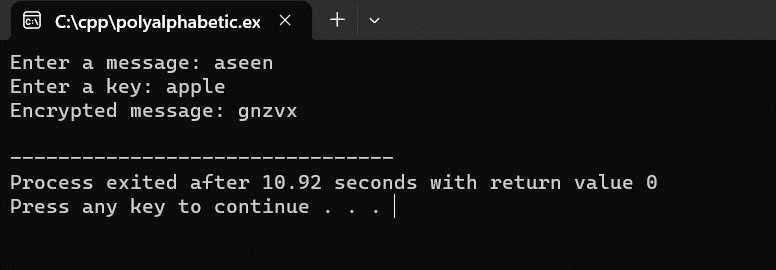
fgets(key, sizeof(key), stdin);

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

polyalphabeticEncrypt(plaintext, key);

return 0;

}



5. Write  
a C program forgeneralization of the Caesar cipher, known as the affine Caesar  
cipher, has thefollowing form: For each plaintext letter p, substitute the  
ciphertextletterC: 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.

Program:

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int modInverse(int a, int m) {

a = a % m;

int x;

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

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

return x;

}

}

return -1;

}

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

if (isalpha(ch)) {

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

return ((a \* (ch - base) + b) % 26) + base;

}

return ch;

}

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

if (isalpha(ch)) {

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

int aInverse = modInverse(a, 26);

if (aInverse == -1) {

printf("The provided key 'a' is not valid for decryption.\n");

return ch;

}

return ((aInverse \* (ch - base - b + 26) % 26) + base);

}

return ch;

}

int main() {

int a, b;

char plaintext[100];

char ciphertext[100];

printf("Enter the value of 'a' (must be relatively prime to 26): ");

scanf("%d", &a);

if (a % 2 == 0 || a % 13 == 0) {

printf("'a' must be relatively prime to 26 for decryption to work.\n");

return 1;

}

printf("Enter the value of 'b': ");

scanf("%d", &b);

printf("Enter the plaintext: ");

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

int i;

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

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

}

ciphertext[i] = '\0';

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

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

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

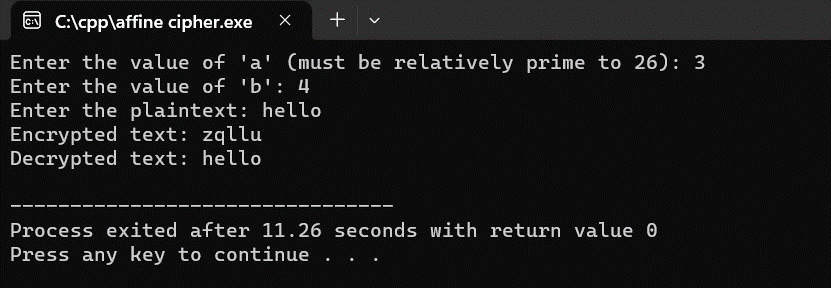
}

plaintext[i] = '\0';

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

return 0;

}



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

Program:

#include <stdio.h>

#include <ctype.h>

#include <string.h>

// Function to calculate the modular multiplicative inverse of a number

int modInverse(int a, int m) {

a = a % m;

int x;

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

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

return x;

}

}

return -1; // No modular inverse exists

}

// Function to decrypt a character using the Affine Caesar cipher

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

if (isalpha(ch)) {

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

int aInverse = modInverse(a, 26);

if (aInverse == -1) {

return ch;

}

return ((aInverse \* (ch - base - b + 26) % 26) + base);

}

return ch;

}

int main() {

char ciphertext[100];

printf("Enter the ciphertext: ");

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

int maxFrequencyB = 0;

int maxFrequencyU = 0;

int bestA = 0;

int bestB = 0;

int a;

int b;

for (a = 1; a < 26; a++) {

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

char plaintext[100]; // Fixed-length array

int frequencyB = 0;

int frequencyU = 0;

int i;

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

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

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

frequencyB++;

} else if (plaintext[i] == 'U') {

frequencyU++;

}

}

plaintext[i] = '\0'; // Null-terminate the plaintext

if (frequencyB > maxFrequencyB && frequencyU > maxFrequencyU) {

maxFrequencyB = frequencyB;

maxFrequencyU = frequencyU;

bestA = a;

bestB = b;

}

}

}

char decryptedText[100]; // Fixed-length array

int i;

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

decryptedText[i] = decrypt(ciphertext[i], bestA, bestB);

}

decryptedText[i] = '\0'; // Null-terminate the decrypted text

printf("Decrypted text with best 'a' and 'b' values (most reasonable result):\n");

printf("a = %d, b = %d\n", bestA, bestB);

printf("%s\n", decryptedText);

return 0;

}

