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.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

#define ALPHABET\_SIZE 26

// Function to generate a random substitution key

void generateRandomKey(char key[]) {

int i, j, used[ALPHABET\_SIZE] = {0};

// Initialize the array with zeros (indicating unused letters)

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

used[i] = 0;

}

// For each position in the key

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

int r;

// Generate a random index that hasn't been used yet

do {

r = rand() % ALPHABET\_SIZE;

} while (used[r]);

// Mark this index as used

used[r] = 1;

// Assign the corresponding letter to the key

key[i] = 'A' + r;

}

// Null-terminate the key string

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

}

// Function to encrypt a message using the monoalphabetic substitution cipher

void encrypt(char message[], const char key[]) {

int i;

char ch;

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

ch = message[i];

if (isupper(ch)) {

// For uppercase letters: Substitute with the corresponding letter in the key

message[i] = key[ch - 'A'];

} else if (islower(ch)) {

// For lowercase letters: First convert to uppercase index, get the substitution,

// then convert back to lowercase

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

}

// Non-alphabetic characters remain unchanged

}

}

// Function to decrypt a message encrypted with the monoalphabetic substitution cipher

void decrypt(char message[], const char key[]) {

int i, j;

char ch;

// To decrypt, we need to find the reverse mapping

char reverse\_key[ALPHABET\_SIZE];

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

// For each letter in the key, find its position in the original alphabet

reverse\_key[key[i] - 'A'] = 'A' + i;

}

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

ch = message[i];

if (isupper(ch)) {

// For uppercase letters: Find the original letter using the reverse key

message[i] = reverse\_key[ch - 'A'];

} else if (islower(ch)) {

// For lowercase letters: Convert to uppercase, find the original letter,

// then convert back to lowercase

message[i] = tolower(reverse\_key[toupper(ch) - 'A']);

}

// Non-alphabetic characters remain unchanged

}

}

// Function to print the substitution mapping

void printKeyMapping(const char key[]) {

printf("Substitution Key Mapping:\n");

printf("Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ\n");

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

}

// Function to create a custom key from a keyword

void createKeyFromKeyword(char key[], const char keyword[]) {

int i, j = 0;

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

char normalized\_keyword[100];

int keyword\_length = 0;

// Normalize the keyword (convert to uppercase and remove duplicates)

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

if (isalpha(keyword[i])) {

char ch = toupper(keyword[i]);

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

used[ch - 'A'] = 1;

normalized\_keyword[keyword\_length++] = ch;

}

}

}

normalized\_keyword[keyword\_length] = '\0';

// Fill the key with the normalized keyword first

strcpy(key, normalized\_keyword);

// Then add the remaining unused letters

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

if (!used[i]) {

key[keyword\_length++] = 'A' + i;

}

}

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

}

int main() {

char message[1000];

char key[ALPHABET\_SIZE + 1]; // +1 for null terminator

int choice, key\_choice;

char keyword[100];

// Seed random number generator

srand(time(NULL));

printf("===== MONOALPHABETIC SUBSTITUTION CIPHER =====\n\n");

// Get user input for key generation

printf("How would you like to generate the substitution key?\n");

printf("1. Random key\n");

printf("2. Keyword-based key\n");

printf("Enter your choice (1/2): ");

scanf("%d", &key\_choice);

getchar(); // Consume the newline

if (key\_choice == 1) {

// Generate a random substitution key

generateRandomKey(key);

} else if (key\_choice == 2) {

// Generate a key from a keyword

printf("Enter a keyword (letters only): ");

fgets(keyword, sizeof(keyword), stdin);

// Remove newline character if present

if (keyword[strlen(keyword) - 1] == '\n')

keyword[strlen(keyword) - 1] = '\0';

createKeyFromKeyword(key, keyword);

} else {

printf("Invalid choice. Using random key by default.\n");

generateRandomKey(key);

}

// Display the key mapping

printKeyMapping(key);

// Get message from user

printf("Enter a message: ");

fgets(message, sizeof(message), stdin);

// Remove newline character if present

if (message[strlen(message) - 1] == '\n')

message[strlen(message) - 1] = '\0';

// Ask user for encryption or decryption

printf("Choose an operation:\n");

printf("1. Encrypt\n");

printf("2. Decrypt\n");

printf("Enter your choice (1/2): ");

scanf("%d", &choice);

// Process based on user's choice

if (choice == 1) {

encrypt(message, key);

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

} else if (choice == 2) {

decrypt(message, key);

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

} else {

printf("Invalid choice. Please run the program again.\n");

return 1;

}

return 0;

}

Output:

