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.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_SIZE 26

// Function to generate a single substitution alphabet using a shift value (Caesar cipher)

void generateSubstitutionAlphabet(int shift, char alphabet[]) {

int i;

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

alphabet[i] = 'A' + ((i + shift) % ALPHABET\_SIZE);

}

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

}

// Function to generate multiple substitution alphabets based on a key

void generateSubstitutionAlphabets(const char \*key, char alphabets[][ALPHABET\_SIZE + 1], int key\_length) {

int i;

for (i = 0; i < key\_length; i++) {

// Calculate the shift value based on the key character's position in the alphabet

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

generateSubstitutionAlphabet(shift, alphabets[i]);

}

}

// Function to encrypt a message using polyalphabetic substitution

void encrypt(const char \*plaintext, char \*ciphertext, char alphabets[][ALPHABET\_SIZE + 1], int key\_length) {

int i, key\_index = 0;

char ch;

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

ch = plaintext[i];

if (isalpha(ch)) {

// Determine which substitution alphabet to use based on the current position in the key

char \*current\_alphabet = alphabets[key\_index];

if (isupper(ch)) {

// For uppercase letters

ciphertext[i] = current\_alphabet[ch - 'A'];

} else {

// For lowercase letters, use uppercase for lookup but preserve case

ciphertext[i] = tolower(current\_alphabet[toupper(ch) - 'A']);

}

// Move to the next key position (wrap around if necessary)

key\_index = (key\_index + 1) % key\_length;

} else {

// Non-alphabetic characters remain unchanged

ciphertext[i] = ch;

}

}

ciphertext[i] = '\0';

}

// Function to decrypt a message encrypted with polyalphabetic substitution

void decrypt(const char \*ciphertext, char \*plaintext, char alphabets[][ALPHABET\_SIZE + 1], int key\_length) {

int i, j, key\_index = 0;

char ch;

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

ch = ciphertext[i];

if (isalpha(ch)) {

// Determine which substitution alphabet to use

char \*current\_alphabet = alphabets[key\_index];

if (isupper(ch)) {

// For uppercase letters: find the position of the ciphertext letter in the alphabet

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

if (current\_alphabet[j] == ch) {

plaintext[i] = 'A' + j;

break;

}

}

} else {

// For lowercase letters

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

if (tolower(current\_alphabet[j]) == ch) {

plaintext[i] = 'a' + j;

break;

}

}

}

// Move to the next key position

key\_index = (key\_index + 1) % key\_length;

} else {

// Non-alphabetic characters remain unchanged

plaintext[i] = ch;

}

}

plaintext[i] = '\0';

}

// Function to display the substitution alphabets

void displaySubstitutionAlphabets(char alphabets[][ALPHABET\_SIZE + 1], int key\_length, const char \*key) {

int i;

printf("\nSubstitution Alphabets:\n");

printf("Standard: ABCDEFGHIJKLMNOPQRSTUVWXYZ\n");

for (i = 0; i < key\_length; i++) {

printf("Key[%c]: %s\n", key[i], alphabets[i]);

}

printf("\n");

}

// Function that implements the Vigenère cipher (a specific type of polyalphabetic cipher)

void vigenereProcess(const char \*input, char \*output, const char \*key, int encrypt) {

int i, key\_index = 0;

int key\_length = strlen(key);

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

if (isalpha(input[i])) {

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

int key\_shift = toupper(key[key\_index]) - 'A';

if (encrypt) {

// Encryption: (plaintext + key) mod 26

output[i] = ((input[i] - base + key\_shift) % ALPHABET\_SIZE) + base;

} else {

// Decryption: (ciphertext - key + 26) mod 26

output[i] = ((input[i] - base - key\_shift + ALPHABET\_SIZE) % ALPHABET\_SIZE) + base;

}

// Move to the next key character

key\_index = (key\_index + 1) % key\_length;

} else {

// Non-alphabetic characters remain unchanged

output[i] = input[i];

}

}

output[i] = '\0';

}

int main() {

char key[100], message[1000], result[1000];

char alphabets[100][ALPHABET\_SIZE + 1]; // Can handle keys up to 100 characters

int choice, method;

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

// Get encryption key from user

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

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

// Remove newline character if present

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

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

// Validate and clean up the key

int valid\_key\_length = 0;

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

if (isalpha(key[i])) {

key[valid\_key\_length++] = toupper(key[i]);

}

}

key[valid\_key\_length] = '\0';

if (valid\_key\_length == 0) {

printf("Error: Key must contain at least one letter.\n");

return 1;

}

// Generate all substitution alphabets based on the key

generateSubstitutionAlphabets(key, alphabets, valid\_key\_length);

// Display the substitution alphabets

displaySubstitutionAlphabets(alphabets, valid\_key\_length, key);

// Ask for implementation method

printf("Choose implementation method:\n");

printf("1. Full polyalphabetic substitution (using separate alphabets)\n");

printf("2. Vigenère cipher (using modular arithmetic)\n");

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

scanf("%d", &method);

getchar(); // Consume the newline

// Get message from user

printf("\nEnter 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("\nChoose 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 and method

if (method == 1) {

if (choice == 1) {

encrypt(message, result, alphabets, valid\_key\_length);

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

} else if (choice == 2) {

decrypt(message, result, alphabets, valid\_key\_length);

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

} else {

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

return 1;

}

} else if (method == 2) {

vigenereProcess(message, result, key, choice == 1);

printf("\n%s message: %s\n", (choice == 1) ? "Encrypted" : "Decrypted", result);

} else {

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

return 1;

}

return 0;

}

Output:

