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.

Code:

/\*\*

\* Implementation of the Affine Caesar Cipher

\* Formula: C = E([a, b], p) = (ap + b) mod 26

\*

\* Requirements:

\* - For decryption to be possible, the cipher must be one-to-one.

\* - This means 'a' must be coprime with 26 (gcd(a, 26) = 1).

\* - 'b' can be any integer, but will be reduced modulo 26.

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

// Function to calculate the greatest common divisor (GCD) using Euclidean algorithm

int gcd(int a, int b) {

int temp;

while (b != 0) {

temp = b;

b = a % b;

a = temp;

}

return a;

}

// Function to calculate modular multiplicative inverse of a under modulo m

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; // No modular inverse exists

}

// Function to check if 'a' is valid for the affine cipher (i.e., coprime with 26)

int isValidA(int a) {

return gcd(a, 26) == 1;

}

// Function to encrypt a single character using the affine cipher

char encryptChar(int a, int b, char p) {

if (!isalpha(p)) {

return p; // Return non-alphabetic characters unchanged

}

// Convert to 0-25 range

int numericValue = isupper(p) ? p - 'A' : p - 'a';

// Apply encryption formula: (a\*p + b) mod 26

int encryptedValue = (a \* numericValue + b) % 26;

// Convert back to ASCII

return isupper(p) ? encryptedValue + 'A' : encryptedValue + 'a';

}

// Function to decrypt a single character using the affine cipher

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

if (!isalpha(c)) {

return c; // Return non-alphabetic characters unchanged

}

// Find modular multiplicative inverse of a

int a\_inverse = modInverse(a, 26);

if (a\_inverse == -1) {

printf("Error: 'a' value has no modular inverse. Decryption impossible.\n");

exit(1);

}

// Convert to 0-25 range

int numericValue = isupper(c) ? c - 'A' : c - 'a';

// Apply decryption formula: a\_inverse \* (c - b) mod 26

int decryptedValue = (a\_inverse \* (numericValue - b + 26)) % 26;

// Convert back to ASCII

return isupper(c) ? decryptedValue + 'A' : decryptedValue + 'a';

}

// Function to encrypt a string using the affine cipher

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

int i;

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

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

}

ciphertext[i] = '\0';

}

// Function to decrypt a string using the affine cipher

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

int i;

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

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

}

plaintext[i] = '\0';

}

// Function to print allowed values of 'a' (those coprime with 26)

void printAllowedAValues() {

printf("Allowed values of 'a' (coprime with 26): ");

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

if (isValidA(a)) {

printf("%d ", a);

}

}

printf("\n");

}

int main() {

int a, b;

char plaintext[1000];

char ciphertext[1000];

char decrypted[1000];

// Print theoretical information about the cipher

printf("==== Affine Caesar Cipher ====\n");

printf("Formula: C = E([a, b], p) = (ap + b) mod 26\n\n");

printf("Limitations on parameters:\n");

printf("- Parameter 'b' can be any integer (will be reduced modulo 26).\n");

printf("- Parameter 'a' must be coprime with 26 (gcd(a, 26) = 1).\n");

printAllowedAValues();

printf("\nDisallowed values of 'a': 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24\n");

printf("These values share common factors with 26, making the cipher not one-to-one.\n\n");

// Get user input

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

scanf("%d", &a);

if (!isValidA(a)) {

printf("Error: 'a' must be coprime with 26. Choose from the allowed values.\n");

return 1;

}

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

scanf("%d", &b);

// Ensure b is in range 0-25

b = ((b % 26) + 26) % 26;

getchar(); // Consume the newline character

printf("Enter the plaintext: ");

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

// Remove trailing newline if present

size\_t len = strlen(plaintext);

if (len > 0 && plaintext[len-1] == '\n') {

plaintext[len-1] = '\0';

}

// Perform encryption

encrypt(a, b, plaintext, ciphertext);

// Perform decryption to verify

decrypt(a, b, ciphertext, decrypted);

// Output results

printf("\nResults:\n");

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

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

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

return 0;

}

Output:

