**CSA5109 - CRYPTOGRAPHY AND NETWORK SECURITY FOR DATA PROTECTION**

**LAB PROGRAMS**

**Date:** 04-11-2023

DAY: SATURDAY

**12.Write a C program for Hill cipher succumbs to a known plaintext attack if sufficient plaintext– ciphertext pairs are provided. It is even easier to solve the Hill cipher if a chosen plaintext attack can be mounted.**

**Program:-**

#include<stdio.h>

#include<string.h>

int main() {

unsigned int a[3][3] = { { 6, 24, 1 }, { 13, 16, 10 }, { 20, 17, 15 } };

unsigned int b[3][3] = { { 8, 5, 10 }, { 21, 8, 21 }, { 21, 12, 8 } };

int i, j;

unsigned int c[20], d[20];

char msg[20];

int determinant = 0, t = 0;

;

printf("Enter plain text\n ");

scanf("%s", msg);

for (i = 0; i < 3; i++) {

c[i] = msg[i] - 65;

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

}

for (i = 0; i < 3; i++) {

t = 0;

for (j = 0; j < 3; j++) {

t = t + (a[i][j] \* c[j]);

}

d[i] = t % 26;

}

printf("\nEncrypted Cipher Text :");

for (i = 0; i < 3; i++)

printf(" %c", d[i] + 65);

for (i = 0; i < 3; i++) {

t = 0;

for (j = 0; j < 3; j++) {

t = t + (b[i][j] \* d[j]);

}

c[i] = t % 26;

}

printf("\nDecrypted Cipher Text :");

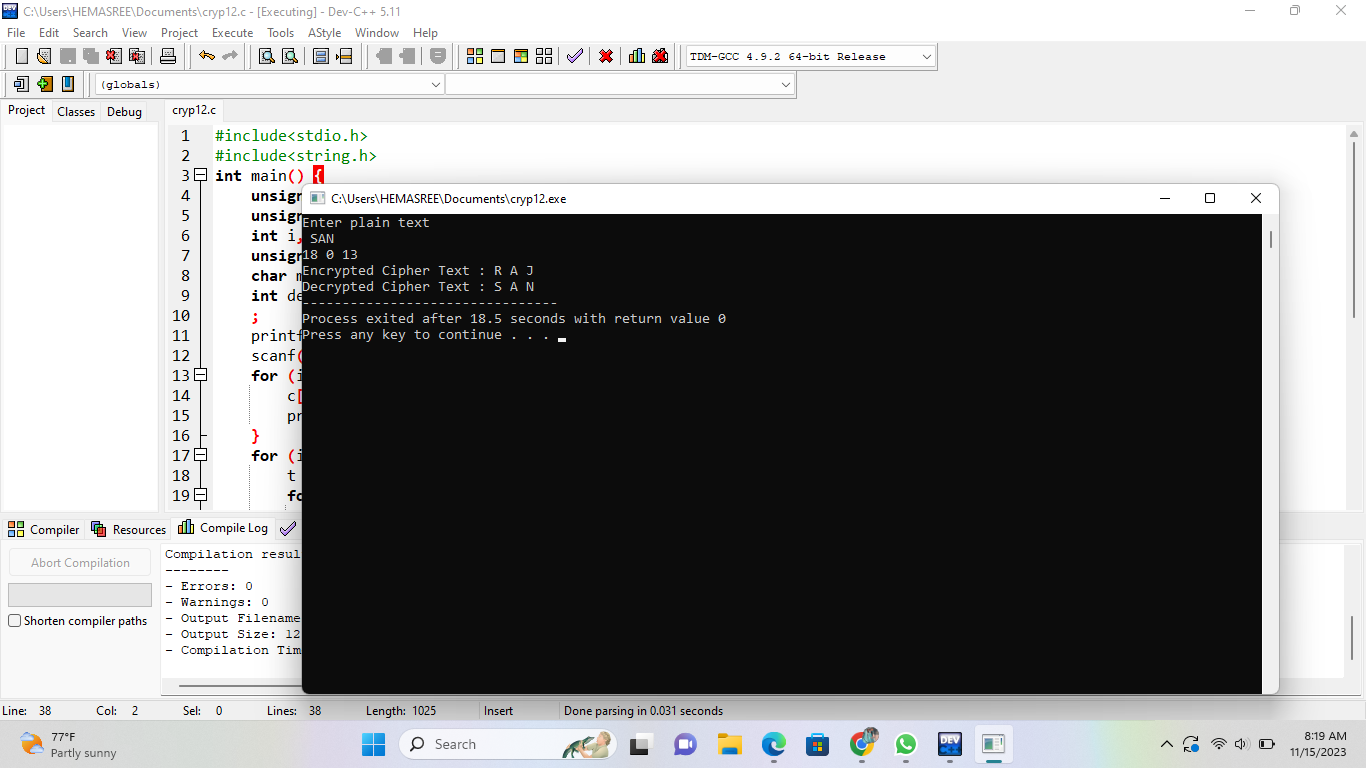
for (i = 0; i < 3; i++)

printf(" %c", c[i] + 65);

return 0;

}

**output:-**

****

**13.** **Write a C program for one-time pad version of the Vigenère cipher. In this scheme, the key is a stream of random numbers between 0 and 26. For example, if the key is 3 19 5 . . . , then the first letter of plaintext is encrypted with a shift of 3 letters, the second with a shift of 19 letters, the third with a shift of 5 letters, and so on.  
a.Encrypt the plaintext send more money with the key stream 9 0 1 7 23 15 21 14 11 11 2 8 9  
b.Using the ciphertext produced in part (a), find a key so that the cipher text decrypts to the plaintext cash not needed.**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

void upper\_case(char \*src) {

while (\*src != '\0') {

if (islower(\*src))

\*src &= ~0x20;

src++;

}

}

char\* encipher(const char \*src, char \*key, int is\_encode) {

int i, klen, slen;

char \*dest;

dest = strdup(src);

upper\_case(dest);

upper\_case(key);

for (i = 0, slen = 0; dest[slen] != '\0'; slen++)

if (isupper(dest[slen]))

dest[i++] = dest[slen];

dest[slen = i] = '\0';

klen = strlen(key);

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

if (!isupper(dest[i]))

continue;

dest[i] = 'A' + (is\_encode ? dest[i] - 'A' + key[i % klen] - 'A'

: dest[i] - key[i % klen] + 26) % 26;

}

return dest;

}

int main() {

const char \*str = "Beware the Jabberwock, my son! The jaws that bite, "

"the claws that catch!";

const char \*cod, \*dec;

char key[] = "VIGENERECIPHER";

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

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

cod = encipher(str, key, 1);

printf("Code: %s\n", cod);

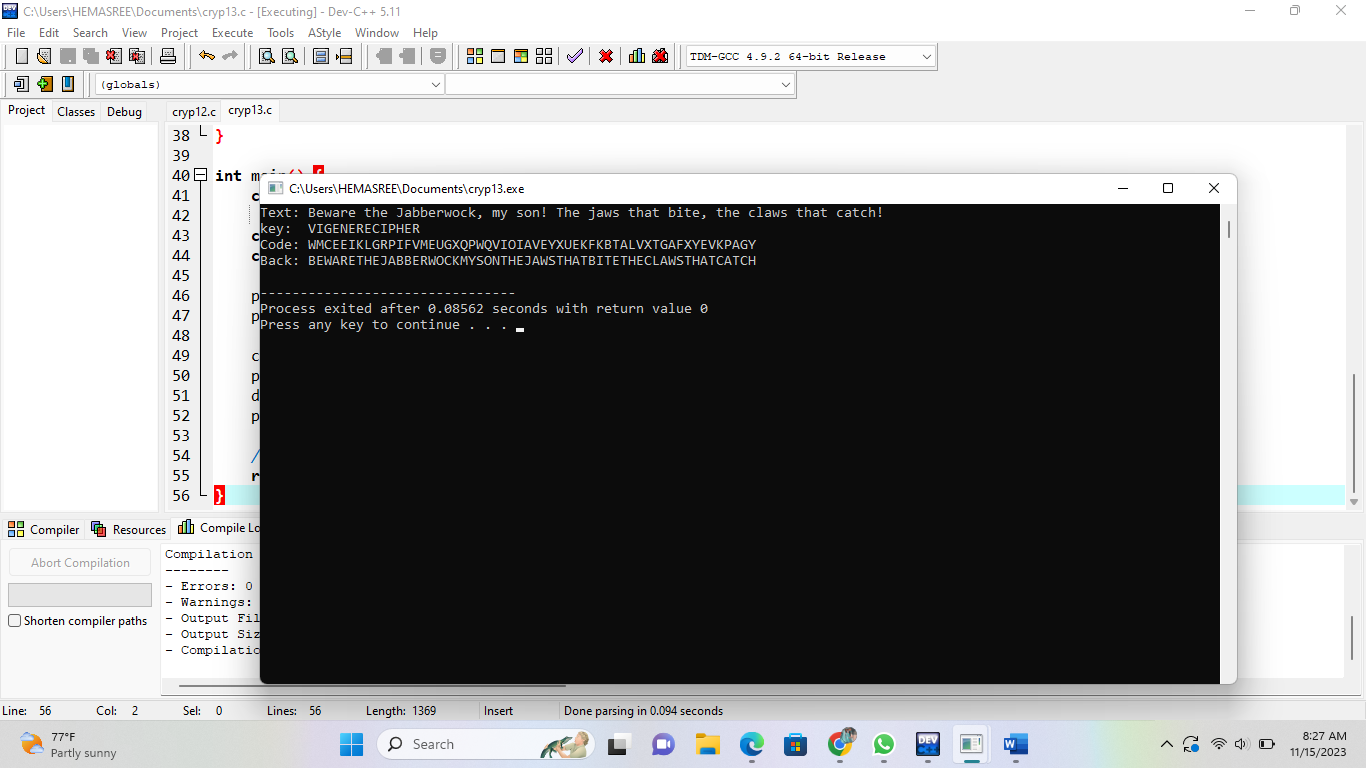
dec = encipher(cod, key, 0);

printf("Back: %s\n", dec);

return 0;

}

**Output:-**

****

**14.** **Write a C program that can perform a letter frequency attack on an additive cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define ALPHABET\_SIZE 26

void calculateFrequency(const char \*text, int \*frequency) {

int i;

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

if (isalpha(text[i])) {

char c = tolower(text[i]);

frequency[c - 'a']++;

}

}

}

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

int len = strlen(ciphertext);

char \*plaintext = (char\*)malloc(len + 1);

int i;

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

if (isalpha(ciphertext[i])) {

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

plaintext[i] = (ciphertext[i] - base - key + ALPHABET\_SIZE) % ALPHABET\_SIZE + base;

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[len] = '\0';

return plaintext;

}

int main() {

char ciphertext[1000];

printf("Enter the ciphertext: ");

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

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

calculateFrequency(ciphertext, frequency);

char mostFrequent = 'e';

int key = (mostFrequent - 'a' + ALPHABET\_SIZE - 1) % ALPHABET\_SIZE;

char \*possiblePlaintext = decrypt(ciphertext, key);

int i;

printf("Top 10 possible plaintexts:\n");

for (i = 0; i < 10; i++) {

printf("%d. Key: %d - %s\n", i + 1, key, possiblePlaintext);

key = (key + 1) % ALPHABET\_SIZE;

strcpy(possiblePlaintext, decrypt(ciphertext, key));

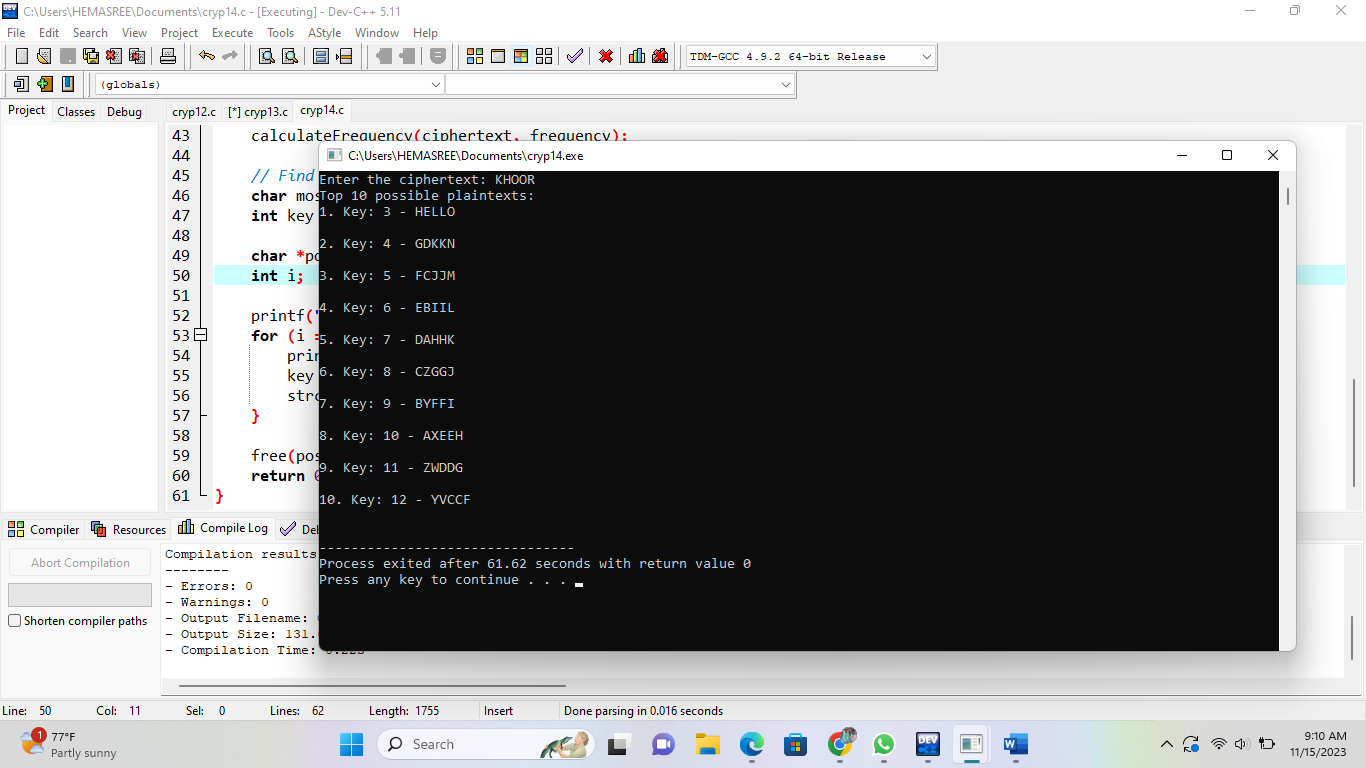
}

free(possiblePlaintext);

return 0;

}

**Output:-**

****

**15. Write a C program that can perform a letter frequency attack on any monoalphabetic substitution cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include <stdbool.h>

#define ALPHABET\_SIZE 26

void calculateFrequencies(char \*text, double \*frequencies) {

int length = strlen(text);

int totalLetters = 0;

int i;

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

if (isalpha(text[i])) {

frequencies[toupper(text[i]) - 'A']++;

totalLetters++;

}

}

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

frequencies[i] /= totalLetters;

frequencies[i] \*= 100;

}

}

double calculateDifference(double \*freq1, double \*freq2) {

double diff = 0.0;

int i;

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

diff += fabs(freq1[i] - freq2[i]);

}

return diff;

}

bool next\_permutation(char \*perm, size\_t size) {

int i = size - 2;

while (i >= 0 && perm[i] >= perm[i + 1]) {

i--;

}

if (i == -1) {

return false;

}

int j = size - 1;

while (perm[j] <= perm[i]) {

j--;

}

char temp = perm[i];

perm[i] = perm[j];

perm[j] = temp;

int left = i + 1;

int right = size - 1;

while (left < right) {

temp = perm[left];

perm[left] = perm[right];

perm[right] = temp;

left++;

right--;

}

return true;

}

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

int length = strlen(ciphertext);

int i;

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

if (isalpha(ciphertext[i])) {

int index = toupper(ciphertext[i]) - 'A';

plaintext[i] = islower(ciphertext[i]) ? tolower(key[index]) : key[index];

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[length] = '\0';

}

int main() {

char ciphertext[1000];

printf("Enter the ciphertext: ");

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

double expectedFrequencies[ALPHABET\_SIZE] = {

8.2, 1.5, 2.8, 4.3, 12.7, 2.2, 2.0, 6.1, 7.0, 0.2, 0.8, 4.0, 2.4, 6.7, 7.5,

1.9, 0.1, 6.0, 6.3, 9.1, 2.8, 1.0, 2.4, 0.2, 2.0, 0.1

};

char key[ALPHABET\_SIZE + 1];

int i;

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

key[i] = 'A' + i;

}

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

char decryptedText[1000];

double ciphertextFrequencies[ALPHABET\_SIZE] = {0};

calculateFrequencies(ciphertext, ciphertextFrequencies);

double minDifference = calculateDifference(ciphertextFrequencies, expectedFrequencies);

do {

decrypt(ciphertext, decryptedText, key);

double currentDifference = calculateDifference(ciphertextFrequencies, expectedFrequencies);

if (currentDifference < minDifference) {

minDifference = currentDifference;

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

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

printf("Difference from expected frequencies: %.2f\n", minDifference);

printf("----------------------------------------\n");

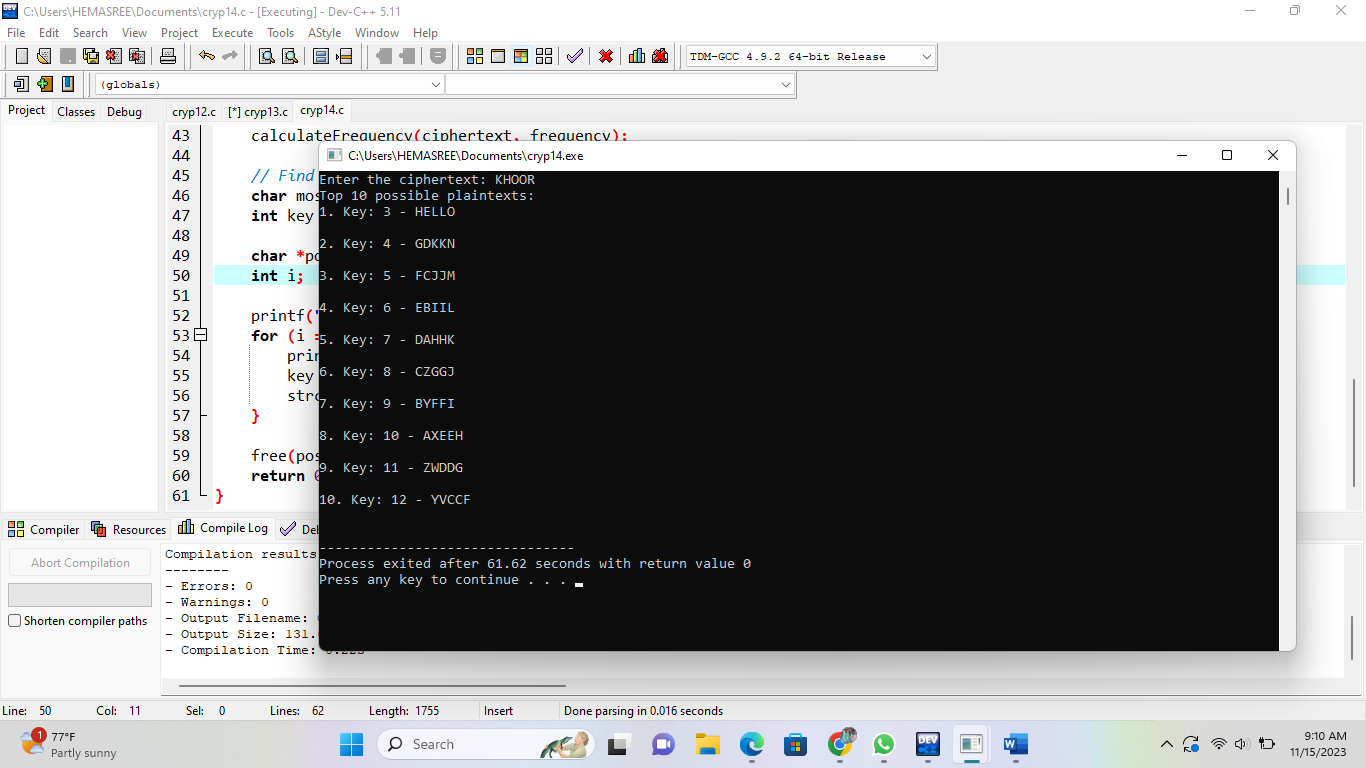
}

} while (next\_permutation(key, ALPHABET\_SIZE));

return 0;

}

**Output:-**



**16**. **Write a python program for DES algorithm for decryption, the 16 keys (K1, K2, c, K16) are used in reverse order. Design a key-generation scheme with the appropriate shift schedule for the decryption process.**

**Program:-**

def encrypt(text,s):

result = ""

for i in range(len(text)):

char = text[i]

if (char.isupper()):

result += chr((ord(char) + s-65) % 26 + 65)

else:

result += chr((ord(char) + s - 97) % 26 + 97)

return result

text = "CEASER CIPHER DEMO"

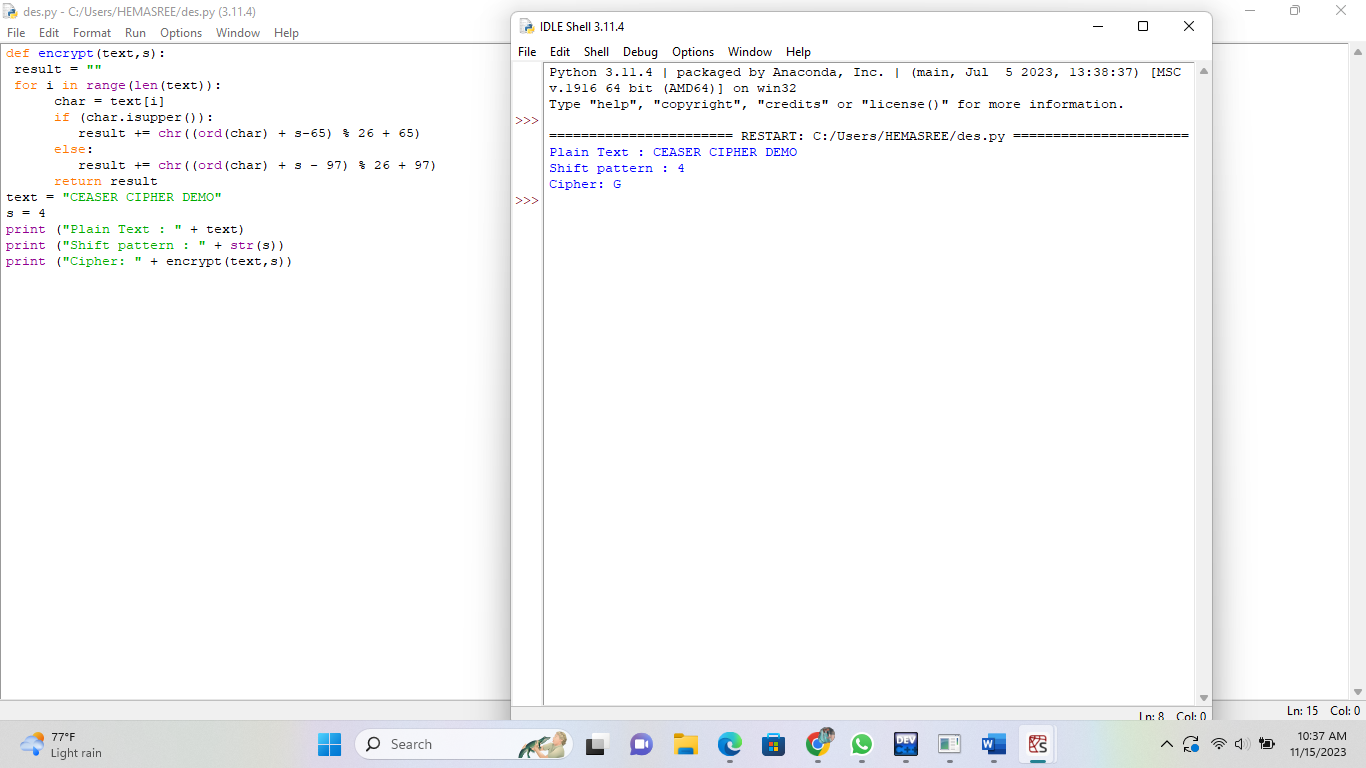
s = 4

print ("Plain Text : " + text)

print ("Shift pattern : " + str(s))

print ("Cipher: " + encrypt(text,s))

**output:-**

****