***SAVEETHA SCHOOL OF ENGINEERING***

***SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCE***

**EXP NO 18: 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.”**

**AIM**

To 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.”

**PROCEDURE**

1. Counts the frequency of each letter in the ciphertext.
2. Sorts the letter frequencies in descending order.
3. Generates a mapping from the ciphertext letters to the plaintext letters based on the frequency analysis.
4. Decrypts the ciphertext using the generated mapping.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define ALPHABET\_SIZE 26

// Structure to store letter frequency information

typedef struct {

char letter;

int frequency;

} LetterFrequency;

// Function to count the frequency of each letter in the ciphertext

void countFrequency(const char \*ciphertext, int frequency[]) {

int i;

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

if (isalpha(ciphertext[i])) {

frequency[toupper(ciphertext[i]) - 'A']++;

}

}

}

// Function to sort letter frequencies in descending order

void sortFrequencies(LetterFrequency frequencies[]) {

int i, j;

LetterFrequency temp;

for (i = 0; i < ALPHABET\_SIZE - 1; i++) {

for (j = 0; j < ALPHABET\_SIZE - 1 - i; j++) {

if (frequencies[j].frequency < frequencies[j + 1].frequency) {

temp = frequencies[j];

frequencies[j] = frequencies[j + 1];

frequencies[j + 1] = temp;

}

}

}

}

// Function to perform monoalphabetic substitution

void decrypt(const char \*ciphertext, const char \*mapping) {

int i;

char plaintext[strlen(ciphertext) + 1];

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

if (isalpha(ciphertext[i])) {

char current = toupper(ciphertext[i]);

plaintext[i] = isupper(mapping[current - 'A']) ? mapping[current - 'A'] : tolower(mapping[current - 'a']);

} else {

plaintext[i] = ciphertext[i];

}

}

plaintext[i] = '\0';

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

}

int main() {

const char \*ciphertext = "CIPHERTEXT\_TO\_DECRYPT";

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

LetterFrequency frequencies[ALPHABET\_SIZE];

char mapping[ALPHABET\_SIZE + 1];

int i;

// Count frequency of letters in ciphertext

countFrequency(ciphertext, frequency);

// Initialize letter frequencies

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

frequencies[i].letter = 'A' + i;

frequencies[i].frequency = frequency[i];

}

// Sort letter frequencies in descending order

sortFrequencies(frequencies);

// Create mapping from ciphertext letters to plaintext letters

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

mapping[frequencies[i].letter - 'A'] = 'A' + i;

}

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

// Decrypt ciphertext using the generated mapping

decrypt(ciphertext, mapping);

return 0;

}

**OUTPUT**

