

40. 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."

Code:

```
import string

from collections import Counter

# English letter frequency (most common → least common)
english_freq_order = "ETAOINSHRDLCLUMWFGYPBVKJXQZ"

# Small dictionary for scoring (can be expanded)
english_words = {
    "THE", "BE", "TO", "OF", "AND", "A", "IN", "THAT", "HAVE", "I", "IT", "FOR", "NOT", "ON",
    "WITH", "HE", "AS", "YOU", "DO", "AT", "THIS", "BUT", "HIS", "BY", "FROM", "THEY", "WE",
    "SAY", "HER", "SHE", "OR", "AN", "WILL", "MY", "ONE", "ALL", "WOULD", "THERE", "THEIR"
}

# Score based on English word matches
def word_score(text):
    words = text.upper().split()
    score = 0
    for w in words:
        if w in english_words:
            score += 5
        if len(w) > 3 and any(w.startswith(e) for e in english_words):
            score += 1
    return score

# Score letter frequency similarity
def frequency_score(text):
    text = text.upper()
    letters = [c for c in text if c in string.ascii_uppercase]
```

```

    freq = Counter(letters)
score = 0
    for i, letter in enumerate(english_freq_order):
        score += freq[letter] * (26 - i)
    return score
# Apply substitution mapping
def apply_mapping(ciphertext, mapping):
    result = ""
    for ch in ciphertext:
        if ch.upper() in mapping:
            new = mapping[ch.upper()]
            result += new.lower() if ch.islower() else new
        else:
            result += ch
    return result
# Generate candidate mappings based on frequency order
def generate_mappings(ciphertext):
    text = ciphertext.upper()
    freq = Counter([c for c in text if c in string.ascii_uppercase])
    cipher_freq_sorted = [x for x, _ in freq.most_common()]
mappings = []
# Initial mapping: highest freq → highest freq
    m = {}
    for i, c in enumerate(cipher_freq_sorted):
        if i < len(english_freq_order):
            m[c] = english_freq_order[i]
    mappings.append(m)

```

```

# Additional variations by swapping nearby frequencies
for shift in range(1, 5): # create ~5 candidate mappings
    m = {}
    for i, c in enumerate(cipher_freq_sorted):
        idx = (i + shift) % len(english_freq_order)
        m[c] = english_freq_order[idx]
    mappings.append(m)
return mappings

# Perform frequency attack
def monoalpha_attack(ciphertext, top_n=10):
    mappings = generate_mappings(ciphertext)
    candidates = []
    for mp in mappings:
        pt = apply_mapping(ciphertext, mp)
        score = frequency_score(pt) + word_score(pt)
        candidates.append((score, pt))

# Sort best candidates
candidates.sort(reverse=True, key=lambda x: x[0])
return candidates[:top_n]

# ----- MAIN -----
ciphertext = input("Enter monoalphabetic substitution ciphertext:\n")
top = int(input("How many top plaintext guesses? "))
results = monoalpha_attack(ciphertext, top)
print("\nTop", top, "possible plaintexts:\n")
for i, (score, text) in enumerate(results, 1):
    print(f"{i}. Score={score}")
    print(text)
    print()

```

```
===== RESTART: C:/Users/Maria/OneDrive/Documents/ex40.py =====
Enter monoalphabetic substitution ciphertext:
Good Morning
How many top plaintext guesses do you want? 6

Top 6 possible plaintexts:

1. Score=262
Teeo Ienasat

2. Score=262
Teeo Neiasat

3. Score=262
Teei Oenasat

4. Score=262
Teei Neoasat

5. Score=262
Teen Oeiasat

6. Score=262
Teen Ieoasat

>> |
```