CRYPTOGRAPHY LAB PROGRAMES -1

MAL REDDY.P

192372015

1) Write a C program for Caesar cipher involves replacing each letter of the alphabet with the letter standing k places further down the alphabet, for k in the range 1 through 25.

```
#include <stdio.h>
#include <ctype.h>
void caesarCipher(char *text, int shift) {
   for (int i = 0; text[i] != '\0'; i++) {
      if (isalpha(text[i])) {
         char base = isupper(text[i]) ? 'A' : 'a';
         text[i] = (text[i] - base + shift) % 26 + base;
      }
   }
}
int main() {
   char text[] = "Hello, World!";
   int shift = 3;
   caesarCipher(text, shift);
   printf("Encrypted: %s\n", text);
   return 0;
}
```

2) Write a C program for monoalphabetic substitution cipher maps a plaintext alphabet to aciphertext alphabet, so that each letter of the plaintext alphabet maps to a single unique letter of the ciphertext alphabet.

```
#include <stdio.h>
#include <string.h>
```

```
#define ALPHABET_SIZE 26
char key[ALPHABET_SIZE] = "QWERTYUIOPASDFGHJKLZXCVBNM";
void encrypt(char *text) {
   for (int i = 0; text[i] != '\0'; i++) {
      if (text[i] >= 'A' && text[i] <= 'Z') {
         text[i] = key[text[i] - 'A'];
      } else if (text[i] >= 'a' && text[i] <= 'z') {
         text[i] = key[text[i] - 'a'] + 32;
      }
   }
}
void decrypt(char *text) {
   for (int i = 0; text[i] != '\0'; i++) {
      if (text[i] >= 'A' && text[i] <= 'Z') {
         for (int j = 0; j < ALPHABET_SIZE; j++) {
            if (key[j] == text[i]) {
               text[i] = 'A' + j;
               break;
            }
         }
      } else if (text[i] >= 'a' && text[i] <= 'z') {
         for (int j = 0; j < ALPHABET_SIZE; j++) {
            if (key[j] == text[i] - 32) {
               text[i] = 'a' + j;
               break;
            }
         }
      }
   }
}
int main() {
   char text[] = "HELLO WORLD";
   encrypt(text);
   printf("Encrypted: %s\n", text);
   decrypt(text);
   printf("Decrypted: %s\n", text);
   return 0;
}
```

3) Write a C program for Playfair algorithm is based on the use of a 5 X 5 matrix of letters constructed using a keyword. Plaintext is encrypted two letters at a time using this matrix.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

char keySquare[SIZE][SIZE];
void prepareKeySquare(char key[]) {
   int map[26] = {0};
   int i, j, k = 0;
   char ch;
   for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++) {
         keySquare[i][j] = '\0';
      }
   }
   for (i = 0; key[i] != '\0'; i++) {
      ch = toupper(key[i]);
      if (ch == 'J') ch = 'I'; // Merge I and J
      if (!map[ch - 'A']) {
         keySquare[k / SIZE][k % SIZE] = ch;
         map[ch - 'A'] = 1;
         k++;
      }
   }
   for (ch = 'A'; ch <= 'Z'; ch++) {
      if (ch == 'J') continue;
      if (!map[ch - 'A']) {
         keySquare[k / SIZE][k % SIZE] = ch;
         map[ch - 'A'] = 1;
         k++;
      }
   }
}
void findPosition(char ch, int *row, int *col) {
   int i, j;
   if (ch == 'J') ch = 'I'; // Merge I and J
   for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++) {
         if (keySquare[i][j] == ch) {
            *row = i;
            *col = j;
            return;
         }
      }
   }
```

```
}
void encryptPair(char a, char b, char *x, char *y) {
    int row1, col1, row2, col2;
    findPosition(a, &row1, &col1);
    findPosition(b, &row2, &col2);
    if (row1 == row2) { // Same row
        *x = keySquare[row1][(col1 + 1) % SIZE];
        *y = keySquare[row2][(col2 + 1) % SIZE];
    } else if (col1 == col2) { // Same column
        *x = keySquare[(row1 + 1) % SIZE][col1];
        *y = keySquare[(row2 + 1) % SIZE][col2];
    } else { // Rectangle rule
        *x = keySquare[row1][col2];
        *y = keySquare[row2][col1];
    }
}
void prepareText(char text[], char newText[]) {
    int len = strlen(text), i, j = 0;
    for (i = 0; i < len; i++) {
        if (isalpha(text[i])) {
            newText[j++] = toupper(text[i]);
        }
    }
    newText[j] = '\0';
    char formatted[100];
    int k = 0;
    for (i = 0; i < j; i++) {
        formatted[k++] = newText[i];
        if (i < j - 1 && newText[i] == newText[i + 1]) {
            formatted[k++] = 'X';
        }
    }
    if (k % 2 != 0) {
        formatted[k++] = 'X';
    }
    formatted[k] = '\0';
    strcpy(newText, formatted);
}
void encryptText(char text[], char encrypted[]) {
    char prepared[100];
    prepareText(text, prepared);
    int i, j = 0;
    for (i = 0; i < strlen(prepared); i += 2) {
        encryptPair(prepared[i], prepared[i + 1], &encrypted[j], &encrypted[j + 1]);
```

```
            j += 2;
        }
        encrypted[j] = '\0';
    }
    int main() {
        char key[100], text[100], encrypted[100];
        printf("Enter the key: ");
        scanf("%s", key);
        prepareKeySquare(key);
        printf("Enter the plaintext: ");
        scanf("%s", text);
        encryptText(text, encrypted);
        printf("Encrypted text: %s\n", encrypted);
        return 0;
    }
```

4) I) As you know, the most frequently occurring letter in English is e. Therefore, the first or second (or perhaps third?) most common character in the message is likely to stand for e. Also, e is often seen in pairs (e.g., meet, fleet, speed, seen, been, agree, etc.). Try to find a character in the ciphertext that decodes to e.
II) The most common word in English is "the." Use this fact to guess the characters that stand for t and h. 3. Decipher the rest of the message by deducing additional words.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 5
void generateMatrix(char *key, char matrix[SIZE][SIZE]) {
    int i, j, k = 0;
    int alphabet[26] = {0};
    for (i = 0; key[i] != '\0'; i++) {
        key[i] = toupper(key[i]);
        if (key[i] == 'J') key[i] = 'I';  // Treat J as I
        if (!alphabet[key[i] - 'A']) {
            alphabet[key[i] - 'A'] = 1;
            matrix[k / SIZE][k % SIZE] = key[i];
            k++;
        }
    }
    for (i = 0; i < 26; i++) {
        if (!alphabet[i] && (i != 'J' - 'A')) {
            matrix[k / SIZE][k % SIZE] = 'A' + i;
```

```
            k++;
        }
    }
}
void printMatrix(char matrix[SIZE][SIZE]) {
    int i, j;
    for (i = 0; i < SIZE; i++) {
        for (j = 0; j < SIZE; j++) {
            printf("%c ", matrix[i][j]);
        }
        printf("\n");
    }
}
void prepareText(char *text, char *preparedText) {
    int i, j = 0;
    for (i = 0; text[i] != '\0'; i++) {
        text[i] = toupper(text[i]);
        if (text[i] == 'J') text[i] = 'I';  // Treat J as I
        if (i < strlen(text) - 1 && text[i] == text[i + 1]) {
            preparedText[j++] = text[i];
            preparedText[j++] = 'X';  // Add filler letter for duplicate letters
            i--;
        } else {
            preparedText[j++] = text[i];
        }
    }
    if (j % 2 != 0) {
        preparedText[j++] = 'X';  // Add filler X for odd length
    }
    preparedText[j] = '\0';
}
void encryptPair(char a, char b, char matrix[SIZE][SIZE], char *encryptedText) {
    int rowA, colA, rowB, colB;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (matrix[i][j] == a) {
                rowA = i;
                colA = j;
            }
            if (matrix[i][j] == b) {
                rowB = i;
                colB = j;
            }
        }
```

```
    }
    if (rowA == rowB) {
        encryptedText[0] = matrix[rowA][(colA + 1) % SIZE];
        encryptedText[1] = matrix[rowB][(colB + 1) % SIZE];
    } else if (colA == colB) {
        encryptedText[0] = matrix[(rowA + 1) % SIZE][colA];
        encryptedText[1] = matrix[(rowB + 1) % SIZE][colB];
    } else {
        encryptedText[0] = matrix[rowA][colB];
        encryptedText[1] = matrix[rowB][colA];
    }
}
void playfairEncrypt(char *plaintext, char *key) {
    char matrix[SIZE][SIZE];
    char preparedText[100];
    char encryptedText[3];
    int i;
    generateMatrix(key, matrix);
    printf("Playfair Matrix:\n");
    printMatrix(matrix);
    printf("\n");
    prepareText(plaintext, preparedText);
    printf("Encrypted Text: ");
        for (i = 0; i < strlen(preparedText); i += 2) {
        encryptPair(preparedText[i], preparedText[i + 1], matrix, encryptedText);
        printf("%c%c", encryptedText[0], encryptedText[1]);
    }
    printf("\n");
}
int main() {
    char plaintext[100], key[100];
    printf("Enter the key (without spaces): ");
    scanf("%s", key);
    printf("Enter the plaintext (without spaces): ");
    scanf("%s", plaintext);
    playfairEncrypt(plaintext, key);
    return 0;
}
```

5) Write a C program for monoalphabetic cipher is that both sender and receiver must commit the permuted cipher sequence to memory. A common technique for avoiding this is to use a keyword from which the cipher sequence can be generated.

For example, using the keyword CIPHER, write out the keyword followed by unused letters in normal order and match this against the plaintext letters.
plain: a b c d e f g h i j k l m n o p q r s t u v w x y z
cipher: C I P H E R A B D F G J K L M N O Q S T U V W X Y Z

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 26
void generateCipherSequence(char *keyword, char *cipherSeq) {
   int i, j = 0;
   int used[SIZE] = {0};

   for (i = 0; keyword[i] != '\0'; i++) {
      char c = toupper(keyword[i]);
      if (isalpha(c) && !used[c - 'A']) {
         cipherSeq[j++] = c;
         used[c - 'A'] = 1;
      }
   }

   for (i = 0; i < SIZE; i++) {
      if (!used[i]) {
         cipherSeq[j++] = 'A' + i;
      }
   }
   cipherSeq[j] = '\0';
}
void encrypt(char *plaintext, char *cipherSeq, char *encryptedText) {
   int i;
   for (i = 0; plaintext[i] != '\0'; i++) {
      if (isalpha(plaintext[i])) {
         char c = toupper(plaintext[i]);
         encryptedText[i] = cipherSeq[c - 'A'];
      } else {
         encryptedText[i] = plaintext[i]; // Non-alphabet characters remain the same
      }
   }
   encryptedText[i] = '\0';
}

void decrypt(char *encryptedText, char *cipherSeq, char *decryptedText) {
   int i;
   for (i = 0; encryptedText[i] != '\0'; i++) {
```

```
        if (isalpha(encryptedText[i])) {
            char c = toupper(encryptedText[i]);
            for (int j = 0; j < SIZE; j++) {
                if (cipherSeq[j] == c) {
                    decryptedText[i] = 'A' + j;
                    break;
                }
            }
        } else {
            decryptedText[i] = encryptedText[i]; // Non-alphabet characters remain the
same
        }
    }
    decryptedText[i] = '\0';
}

int main() {
    char plaintext[100], keyword[100], encryptedText[100], decryptedText[100];
    char cipherSeq[SIZE + 1];

    printf("Enter the keyword: ");
    scanf("%s", keyword);


    generateCipherSequence(keyword, cipherSeq);

    printf("Generated Cipher Sequence: %s\n", cipherSeq);

    printf("Enter the plaintext: ");
    scanf("%s", plaintext);


    encrypt(plaintext, cipherSeq, encryptedText);
    printf("Encrypted Text: %s\n", encryptedText);
    decrypt(encryptedText, cipherSeq, decryptedText);
    printf("Decrypted Text: %s\n", decryptedText);
    return 0;
}
```

6) Write a C program for Playfair matrix:

M F H I/J K
U N O P Q
Z V W X Y
E L A R G
D S T B C

Encrypt this message: Must see you over Cadogan West. Coming at once.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 2
#define MOD 26
int determinant(int matrix[SIZE][SIZE]) {
   return (matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]) % MOD;
}
int modInverse(int a, int m) {
   for (int x = 1; x < m; x++) {
      if ((a * x) % m == 1)
         return x;
   }
   return -1;
}
int inverseMatrix(int matrix[SIZE][SIZE], int invMatrix[SIZE][SIZE]) {
   int det = determinant(matrix);
   int detInverse = modInverse(det, MOD);
   if (detInverse == -1) {
      return 0;
   }
   invMatrix[0][0] = matrix[1][1] * detInverse % MOD;
   invMatrix[0][1] = (-matrix[0][1]) * detInverse % MOD;
   invMatrix[1][0] = (-matrix[1][0]) * detInverse % MOD;
   invMatrix[1][1] = matrix[0][0] * detInverse % MOD;
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         if (invMatrix[i][j] < 0) invMatrix[i][j] += MOD;
      }
   }

   return 1;
}
void matrixMultiply(int matrix[SIZE][SIZE], int vec[SIZE], int result[SIZE]) {
```

```
    for (int i = 0; i < SIZE; i++) {
        result[i] = 0;
        for (int j = 0; j < SIZE; j++) {
            result[i] = (result[i] + matrix[i][j] * vec[j]) % MOD;
        }
    }
}
int charToNum(char c) {
    return tolower(c) - 'a';
}
char numToChar(int num) {
    return (num + 'a');
}
void hillEncrypt(char *plaintext, int keyMatrix[SIZE][SIZE], char *ciphertext) {
    int len = strlen(plaintext);
    int vec[SIZE], result[SIZE];
        for (int i = 0, j = 0; i < len; i++) {
        if (isalpha(plaintext[i])) {
            vec[0] = charToNum(plaintext[i]);
            if (i + 1 < len && isalpha(plaintext[i + 1])) {
                vec[1] = charToNum(plaintext[i + 1]);
                i++;
            } else {
                vec[1] = charToNum('x');
            }
            matrixMultiply(keyMatrix, vec, result);
            ciphertext[j++] = numToChar(result[0]);
            ciphertext[j++] = numToChar(result[1]);
        }
    }
    ciphertext[len] = '\0';
}
void hillDecrypt(char *ciphertext, int invKeyMatrix[SIZE][SIZE], char *plaintext) {
    int len = strlen(ciphertext);
    int vec[SIZE], result[SIZE];
        for (int i = 0, j = 0; i < len; i++) {
        if (isalpha(ciphertext[i])) {
            vec[0] = charToNum(ciphertext[i]);
            vec[1] = charToNum(ciphertext[i + 1]);
            matrixMultiply(invKeyMatrix, vec, result);
            plaintext[j++] = numToChar(result[0]);
            plaintext[j++] = numToChar(result[1]);
            i++;
        }
```

```
    }
    plaintext[len] = '\0';
}
int main() {
    char plaintext[] = "meet me at the usual place at ten rather than eight oclock";
    char ciphertext[100], decryptedText[100];
    int keyMatrix[SIZE][SIZE] = {{9, 4}, {5, 7}};
    int invKeyMatrix[SIZE][SIZE];
    hillEncrypt(plaintext, keyMatrix, ciphertext);
    printf("Ciphertext: %s\n", ciphertext);
    if (inverseMatrix(keyMatrix, invKeyMatrix)) {
        hillDecrypt(ciphertext, invKeyMatrix, decryptedText);
        printf("Decrypted Text: %s\n", decryptedText);
    } else {
        printf("Decryption failed\n");
    }
    return 0;
}
```

7) Write a C program to Encrypt the message “meet me at the usual place at ten rather than eight oclock” using the Hill cipher with the key.
(9 4)
(5 7)
a. Show your calculations and the result.
b. Show the calculations for the corresponding decryption of the ciphertext to recover the original plaintext

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define SIZE 5
char matrix[SIZE][SIZE] = {
    {'M', 'F', 'H', 'I', 'J'},
    {'K', 'U', 'N', 'O', 'P'},
    {'Q', 'Z', 'V', 'W', 'X'},
    {'Y', 'E', 'L', 'A', 'R'},
    {'G', 'D', 'S', 'T', 'B'}
};
void preprocessText(char* text, char* processedText) {
    int j = 0;
    for (int i = 0; text[i] != '\0'; i++) {
        if (isalpha(text[i])) {
            processedText[j++] = toupper(text[i]);
        }
```

```
    }
    processedText[j] = '\0';
}
void preprocessPairs(char* text, char* processedText) {
    int len = strlen(text);
    for (int i = 0, j = 0; i < len; i++) {
        if (text[i] == 'J') {
            processedText[j++] = 'I';
        } else {
            processedText[j++] = text[i];
        }
    }
    processedText[len] = '\0';
}
void findPosition(char ch, int* row, int* col) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (matrix[i][j] == ch) {
                *row = i;
                *col = j;
                return;
            }
        }
    }
}
void playfairEncrypt(char* plaintext, char* ciphertext) {
    int len = strlen(plaintext);
    if (len % 2 != 0) {
        plaintext[len++] = 'X';
    }
    int i = 0;
    char pair[2];
    int row1, col1, row2, col2;
```