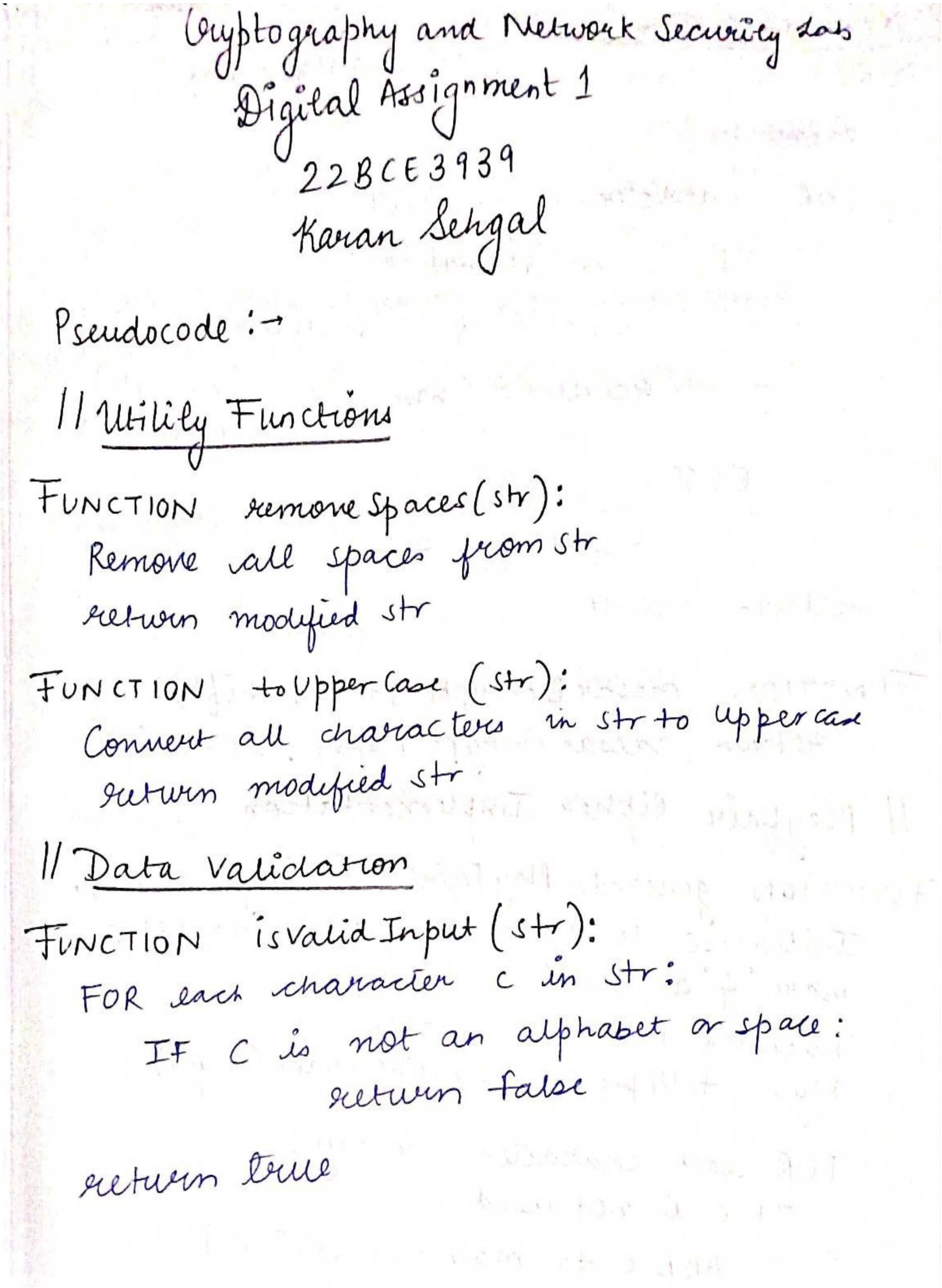
Cryptography and Network Security Lab
Digital Assignment 1
22BCE3939
Karan Sehgal

Using necessary user defined functions and proper data validation develop a generic menu driven code to simulate the working of the following conventional ciphers. i. Caesar Cipher i. Play fair Cipher iii. Hill Cipher

Pseudocode:



Cryptography and Network Security Lab

Digital Assignment 1

22BCE3939

Karan Sehgal

Pseudocode :→

```
// Utility Functions

FUNCTION removeSpaces(str):
    Remove all spaces from str
    return modified str

FUNCTION toUpperCase (str):
    Convert all characters in str to uppercase
    return modified str

// Data Validation

FUNCTION isValidInput (str):
    FOR each character c in str:
        IF c is not an alphabet or space:
            return false

    return true
```

```
// Caesar Cipher Functions        22BCE3939
FUNCTION caesarEncrypt (text, shift):
    result = ""
    FOR each character c in text:
        IF c is alphabet:
            base = 'A' if c is upper else 'a'
            result += (base + (c - base + shift)
                                          MOD 26 )
        ELSE:
            result += c
    return result

FUNCTION caesarDecrypt (text, shift):
    return caesarEncrypt (text, 26 - shift)
// Playfair Cipher Implementation
FUNCTION generatePlayFairMatrix (key, matrix):
    Initialise used array of 26 size to false
    mare 'j' as used
    Initialise row = 0, col = 0
    key = toUpperCase (remone space (key))
    FOR each character c in key:
        IF c is not used
            ADD c to matrix [row][col]
            Mark c used
            Inc col
            IF col == 5 , reset col and Inc row

    Repeat with all alphabets and fill
    up the matrix
```

22BCE3939

```
FUNCTION findPosition (matrix, c, row, col):
    Replace 'J' with 'I' in c
    FOR each position (i, j) in matrix:
        IF matrix[i][j] == c :
            row = i , col = j
                return {i, j}

    return {-1, -1}

FUNCTION playFair Encrypt (text, key):
    Generate playfair matrix
    text = touppercase (removespaces ( text))
    IF text length is odd, append 'x'
    initialise result = " "

    FOR i=0 to text.length with step 2:
        FIND positions of text[i] and text[i+1]
        IF same row:
            Add right neighbor to result
        ELSE IF same column:
            Add below neighbor to result
        ELSE:
            Add opposite corner characters
            when formed a rectangle to
            result.

    Return result
```

```
FUNCTION playfairDecrypt (text, key):
    Generate Playfair matrix
    initialise result = " "

    FOR i=0 to text.length with step 2:
        Find positions of text[i] and text[i+1]
          in matrix:
        IF same row:
            Add left neighbor to result
        ELSE IF same column:
            Add above neighbor to result
        ELSE:
            Add opposite corner characters
                to result

    Return result
```

// Hill Cipher Implementation

```
FUNCTION modInverse (a):
    FOR i=1 to 25:
        IF (a*i) MOD 26 == 1:
            return i
    return -1
```

22BCE3939

```
FUNCTION adjoint (matrix, adj):
    compute cofactor matrix
    fill adjoint matrix

FUNCTION determinant (matrix, n):
    IF (n == 1):
        return matrix[0][0]
    Initialise D = 0
    FOR f = 0 to n-1:
        Compute cofactor matrix and
            recursive determinant
        Add signed value to D
    return D

FUNCTION getKeyMatrix (key, size):
    IF key.length() != size * size:
        THROW error
    Fill KeyMatrix with values in
        the key string
    Return KeyMatrix
```

22BCE3939

```
FUNCTION hillEncrypt (text, key, size):
    Generate keyMatrix
    Pad text with 'x' to make length
     a multiple of size.
    Initialise result = " "

    FOR each block of size in text:
        Compute matrix multiplication &
            modulo 26
        Append result characters
    Return result
FUNCTION hillDecrypt (text, key, size):
    Generate Key Matrix
    Compute determinant & its modular
     inverse
    Compute adjoint and inverse key matrix
    Initialize result = " "
    FOR each block of size in text:
        Compute matrix multiplication with
        inverse key matrix and modulo 26
        Append result characters
    Return result
```

```
// Menu Driven Driver Code                //22BCE3939
PROCEDURE main()
    Repeat
        Display menu Options
        Read choice

        If choice is valid
            Read text
            Validate text

            CASE choice OF
                1: DO Caesar Encrypt
                2: DO Caesar Decrypt
                3: DO PlayFair Encrypt
                4: DO PlayFair Decrypt
                5: Do Hill Encrypt
                    Read matrix size
                    VALIDATE size
                    Read key of length $size^2$
                6: Do Hill Decrypt
                    similar to Case 5

            ENDCASE

            Display result
    UNTIL choice is exit
```

Source Code:

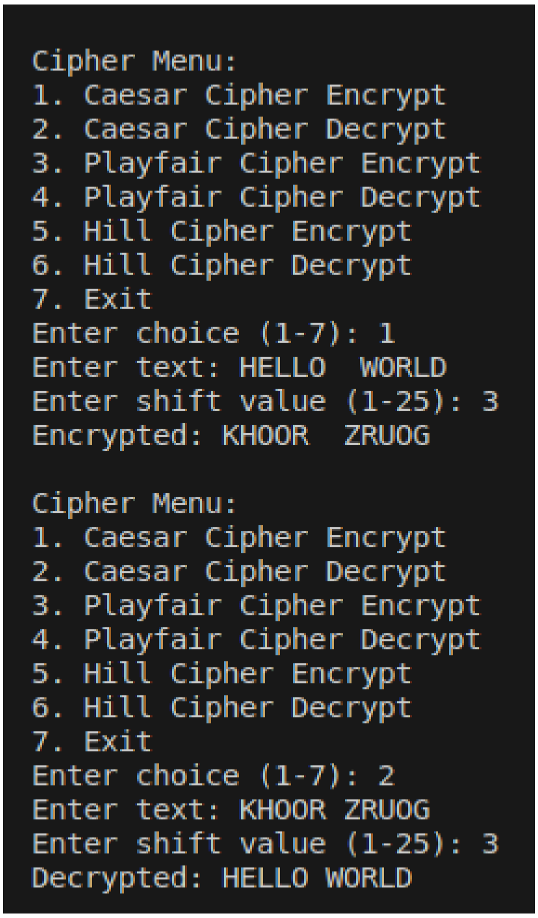
```
//22BCE3939
//Karan Sehgal
//DA1 Implementation of Caesar, Playfair and Hill Cipher in C++
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <cctype>
#include <cmath>
using namespace std;

// Utility functions remain the same
string removeSpaces(string str) {
    str.erase(remove(str.begin(), str.end(), ' '), str.end());
    return str;
}

string toUpperCase(string str) {
    transform(str.begin(), str.end(), str.begin(), ::toupper);
    return str;
}

bool isValidInput(string str) {
    return all_of(str.begin(), str.end(), [](char c) {
        return isalpha(c) || isspace(c);
    });
}

// Caesar Cipher functions
string caesarEncrypt(string text, int shift) {
    string result = "";
    for(char c : text) {
        if(isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            result += char(base + (c - base + shift) % 26);
        } else {
            result += c;
        }
    }
    return result;
}
```

```
string caesarDecrypt(string text, int shift) {
    return caesarEncrypt(text, 26 - shift);
}

// Enhanced Playfair Cipher functions
void generatePlayfairMatrix(string key, char matrix[5][5]) {
    bool used[26] = {false};
    used['J' - 'A'] = true;

    int row = 0, col = 0;
    key = toUpperCase(removeSpaces(key));

    for(char c : key) {
        if(!used[c - 'A']) {
            matrix[row][col] = c;
            used[c - 'A'] = true;
            col++;
            if(col == 5) {
                col = 0;
                row++;
            }
        }
    }

    for(char c = 'A'; c <= 'Z'; c++) {
        if(!used[c - 'A']) {
            matrix[row][col] = c;
            col++;
            if(col == 5) {
                col = 0;
                row++;
            }
        }
    }
}

void findPosition(char matrix[5][5], char c, int& row, int& col) {
    if(c == 'J') c = 'I';
    for(int i = 0; i < 5; i++)
        for(int j = 0; j < 5; j++)
            if(matrix[i][j] == c) {
                row = i;
                col = j;
                return;
            }
}

string playfairEncrypt(string text, string key) {
    char matrix[5][5];
    generatePlayfairMatrix(key, matrix);

    text = toUpperCase(removeSpaces(text));
    if(text.length() % 2 != 0) text += 'X';

    string result = "";
    for(size_t i = 0; i < text.length(); i += 2) {
        int row1, col1, row2, col2;
        findPosition(matrix, text[i], row1, col1);
        findPosition(matrix, text[i+1], row2, col2);

        if(row1 == row2) {
            result += matrix[row1][(col1 + 1) % 5];
            result += matrix[row2][(col2 + 1) % 5];
        }
        else if(col1 == col2) {
            result += matrix[(row1 + 1) % 5][col1];
            result += matrix[(row2 + 1) % 5][col2];
        }
        else {
            result += matrix[row1][col2];
            result += matrix[row2][col1];
        }
    }
    return result;
}

string playfairDecrypt(string text, string key) {
    char matrix[5][5];
    generatePlayfairMatrix(key, matrix);

    string result = "";
    for(size_t i = 0; i < text.length(); i += 2) {
        int row1, col1, row2, col2;
        findPosition(matrix, text[i], row1, col1);
        findPosition(matrix, text[i+1], row2, col2);

        if(row1 == row2) {
            result += matrix[row1][(col1 + 4) % 5];
            result += matrix[row2][(col2 + 4) % 5];
        }
        else if(col1 == col2) {
            result += matrix[(row1 + 4) % 5][col1];
            result += matrix[(row2 + 4) % 5][col2];
        }
        else {
            result += matrix[row1][col2];
            result += matrix[row2][col1];
        }
    }
    return result;
}

// Enhanced Hill Cipher functions
int modInverse(int a) {
    for(int i = 1; i < 26; i++)
        if(((a % 26) * (i % 26)) % 26 == 1)
            return i;
    return -1;
}

void getCofactor(vector<vector<int>>& matrix, vector<vector<int>>& temp, int p, int
q, int n) {
    int i = 0, j = 0;
    for(int row = 0; row < n; row++) {
        for(int col = 0; col < n; col++) {
            if(row != p && col != q) {
                temp[i][j++] = matrix[row][col];
                if(j == n - 1) {
                    j = 0;
                    i++;
                }
            }
        }
    }
}

int determinant(vector<vector<int>>& matrix, int n) {
    if(n == 1) return matrix[0][0];
    int D = 0;
    vector<vector<int>> temp(n, vector<int>(n));
    int sign = 1;
    for(int f = 0; f < n; f++) {
        getCofactor(matrix, temp, 0, f, n);
        D += sign * matrix[0][f] * determinant(temp, n-1);
        sign = -sign;
    }
    return D;
}

void adjoint(vector<vector<int>>& matrix, vector<vector<int>>& adj) {
    int N = matrix.size();
    if(N == 1) {
        adj[0][0] = 1;
        return;
    }
    int sign = 1;
    vector<vector<int>> temp(N, vector<int>(N));

    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            getCofactor(matrix, temp, i, j, N);
            sign = ((i+j) % 2 == 0)? 1: -1;
            adj[j][i] = (sign) * (determinant(temp, N-1));
            adj[j][i] = ((adj[j][i] % 26) + 26) % 26;
        }
    }
}

void getKeyMatrix(string key, vector<vector<int>>& keyMatrix, int size) {
    if (key.length() != size * size) {
        throw runtime_error("Key length must be " + to_string(size * size));
    }
    int k = 0;
    for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
            keyMatrix[i][j] = (key[k++] - 'A') % 26;
}

string hillEncrypt(string text, string key, int size) {
    vector<vector<int>> keyMatrix(size, vector<int>(size));
    getKeyMatrix(key, keyMatrix, size);

    while(text.length() % size != 0)
        text += 'X';

    string result = "";
    for(size_t i = 0; i < text.length(); i += size) {
        for(int j = 0; j < size; j++) {
            int sum = 0;
            for(int k = 0; k < size; k++) {
                sum += keyMatrix[j][k] * (text[i + k] - 'A');
            }
            result += char((sum % 26) + 'A');
        }
    }
    return result;
}

string hillDecrypt(string text, string key, int size) {
    vector<vector<int>> keyMatrix(size, vector<int>(size));
    getKeyMatrix(key, keyMatrix, size);

    int det = determinant(keyMatrix, size);
    det = ((det % 26) + 26) % 26;
    int detInv = modInverse(det);

    if(detInv == -1) {
        return "Invalid key: inverse doesn't exist!";
    }

    vector<vector<int>> adj(size, vector<int>(size));
    adjoint(keyMatrix, adj);

    for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
            keyMatrix[i][j] = (detInv * adj[i][j]) % 26;

    string result = "";
    for(size_t i = 0; i < text.length(); i += size) {
        for(int j = 0; j < size; j++) {
            int sum = 0;
            for(int k = 0; k < size; k++) {
                sum += keyMatrix[j][k] * (text[i + k] - 'A');
            }
            result += char(((sum % 26) + 26) % 26 + 'A');
        }
    }
    return result;
}
```

```
int main() {
    int choice;
    string text, key;

    do {
        cout << "\nCipher Menu:\n";
        cout << "1. Caesar Cipher Encrypt\n";
        cout << "2. Caesar Cipher Decrypt\n";
        cout << "3. Playfair Cipher Encrypt\n";
        cout << "4. Playfair Cipher Decrypt\n";
        cout << "5. Hill Cipher Encrypt\n";
        cout << "6. Hill Cipher Decrypt\n";
        cout << "7. Exit\n";
        cout << "Enter choice (1-7): ";
        cin >> choice;
        cin.ignore();

        if(choice >= 1 && choice <= 6) {
            cout << "Enter text: ";
            getline(cin, text);

            if(!isValidInput(text)) {
                cout << "Invalid input! Use only alphabets and spaces.\n";
                continue;
            }


            switch(choice) {
                case 1: {
                    int shift;
                    cout << "Enter shift value (1-25): ";
                    cin >> shift;
                    if(shift < 1 || shift > 25) {
                        cout << "Invalid shift value!\n";
                        break;
                    }
                    cout << "Encrypted: " << caesarEncrypt(text, shift) << endl;
                    break;
                }
                case 2: {
                    int shift;
                    cout << "Enter shift value (1-25): ";
                    cin >> shift;
                    if(shift < 1 || shift > 25) {
                        cout << "Invalid shift value!\n";
                        break;
                    }
                    cout << "Decrypted: " << caesarDecrypt(text, shift) << endl;
                    break;
                }
                case 3: {
                    cout << "Enter key: ";
                    cin >> key;
                    if(!isValidInput(key)) {
                        cout << "Invalid key! Use only alphabets.\n";
                        break;
                    }
                    cout << "Encrypted: " << playfairEncrypt(text, key) << endl;
                    break;
                }
                case 4: {
                    cout << "Enter key: ";
                    cin >> key;
                    if(!isValidInput(key)) {
                        cout << "Invalid key! Use only alphabets.\n";
                        break;
                    }
                    cout << "Decrypted: " << playfairDecrypt(text, key) << endl;
                    break;
                }
                case 5: {
                        int size;
                        cout << "Enter matrix size (n): ";
                        cin >> size;
                        cout << "Enter key (" << size * size << " letters): ";
                        cin >> key;
                        if(!isValidInput(key)) {
                            cout << "Invalid key! Use only alphabets.\n";
                            break;
                        }
                        key = toUpperCase(key);
                        cout << "Encrypted: " << hillEncrypt(toUpperCase(removeSpaces(text)),
toUpperCase(key), size) << endl;
                        break;
                    }
                case 6: {
                        int size;
                        cout << "Enter matrix size (n): ";
                        cin >> size;
                        cout << "Enter key (" << size * size << " letters): ";
                        cin >> key;
                        if(!isValidInput(key)) {
                            cout << "Invalid key! Use only alphabets.\n";
                            break;
                        }
                        key = toUpperCase(key);
                        cout << "Decrypted: " << 
hillDecrypt(toUpperCase(removeSpaces(text)),toUpperCase(key), size) << endl;
                        break;
                    }
            }
        }
    } while(choice != 7);

    return 0;
}
```

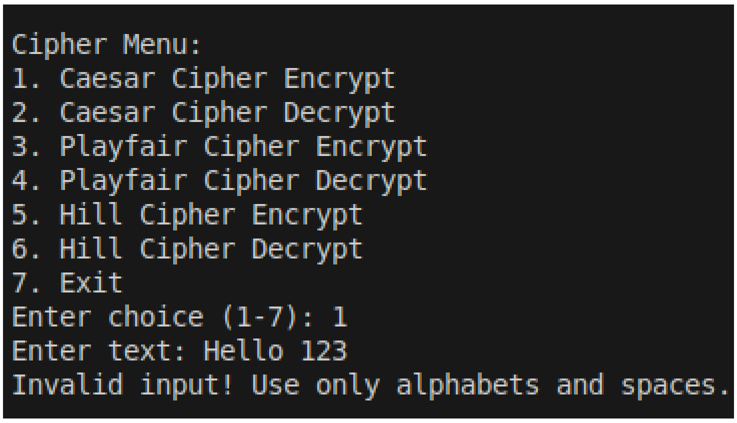
Output:

Caesar Cipher :



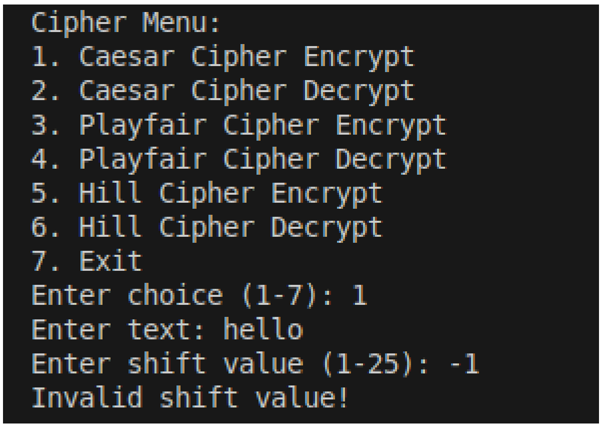
```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 1
Enter text: HELLO  WORLD
Enter shift value (1-25): 3
Encrypted: KHOOR  ZRUOG

Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 2
Enter text: KHOOR ZRUOG
Enter shift value (1-25): 3
Decrypted: HELLO WORLD
```

Only allows alphabets and spaces:



```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 1
Enter text: Hello 123
Invalid input! Use only alphabets and spaces.
```

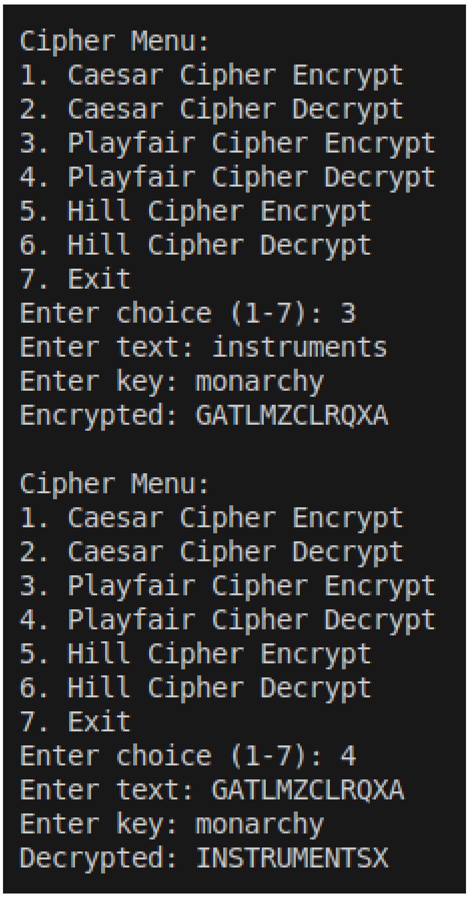
Only Valid shifts allowed



```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 1
Enter text: hello
Enter shift value (1-25): -1
Invalid shift value!
```

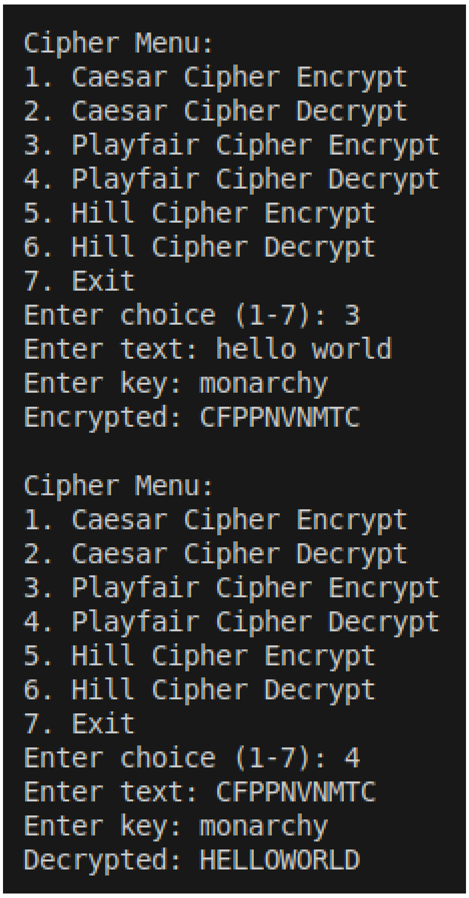
Playfair Cipher:

1) Encrypting words with no repeating letters:



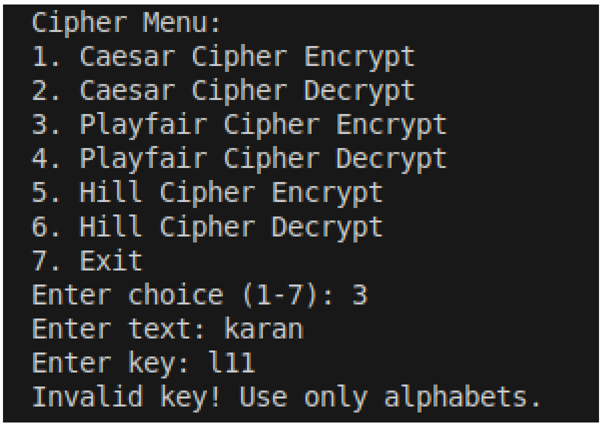
```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 3
Enter text: instruments
Enter key: monarchy
Encrypted: GATLMZCLRQXA

Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 4
Enter text: GATLMZCLRQXA
Enter key: monarchy
Decrypted: INSTRUMENTSX
```

2) Encrypting words with repeating letters:



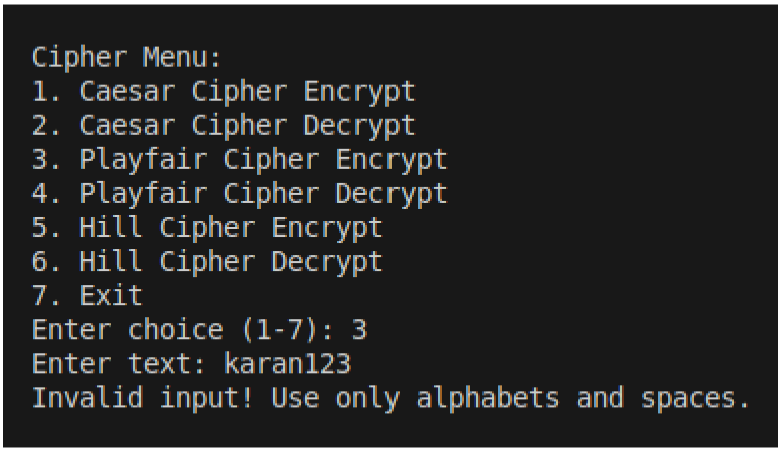
```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 3
Enter text: hello world
Enter key: monarchy
Encrypted: CFPPNVNMTC

Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 4
Enter text: CFPPNVNMTC
Enter key: monarchy
Decrypted: HELLOWORLD
```

Invalid key:



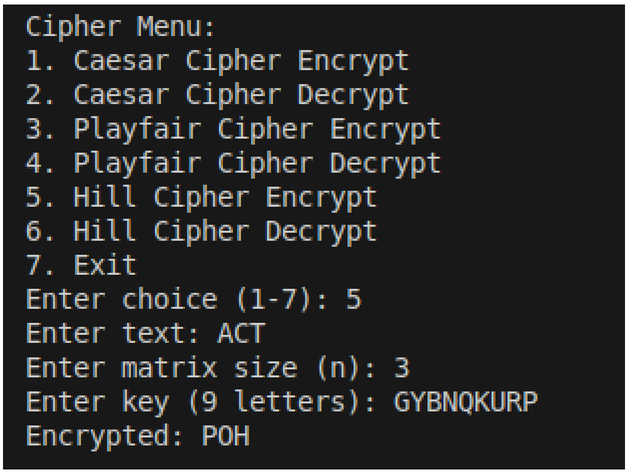
```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 3
Enter text: karan
Enter key: l11
Invalid key! Use only alphabets.
```

Invalid Text:

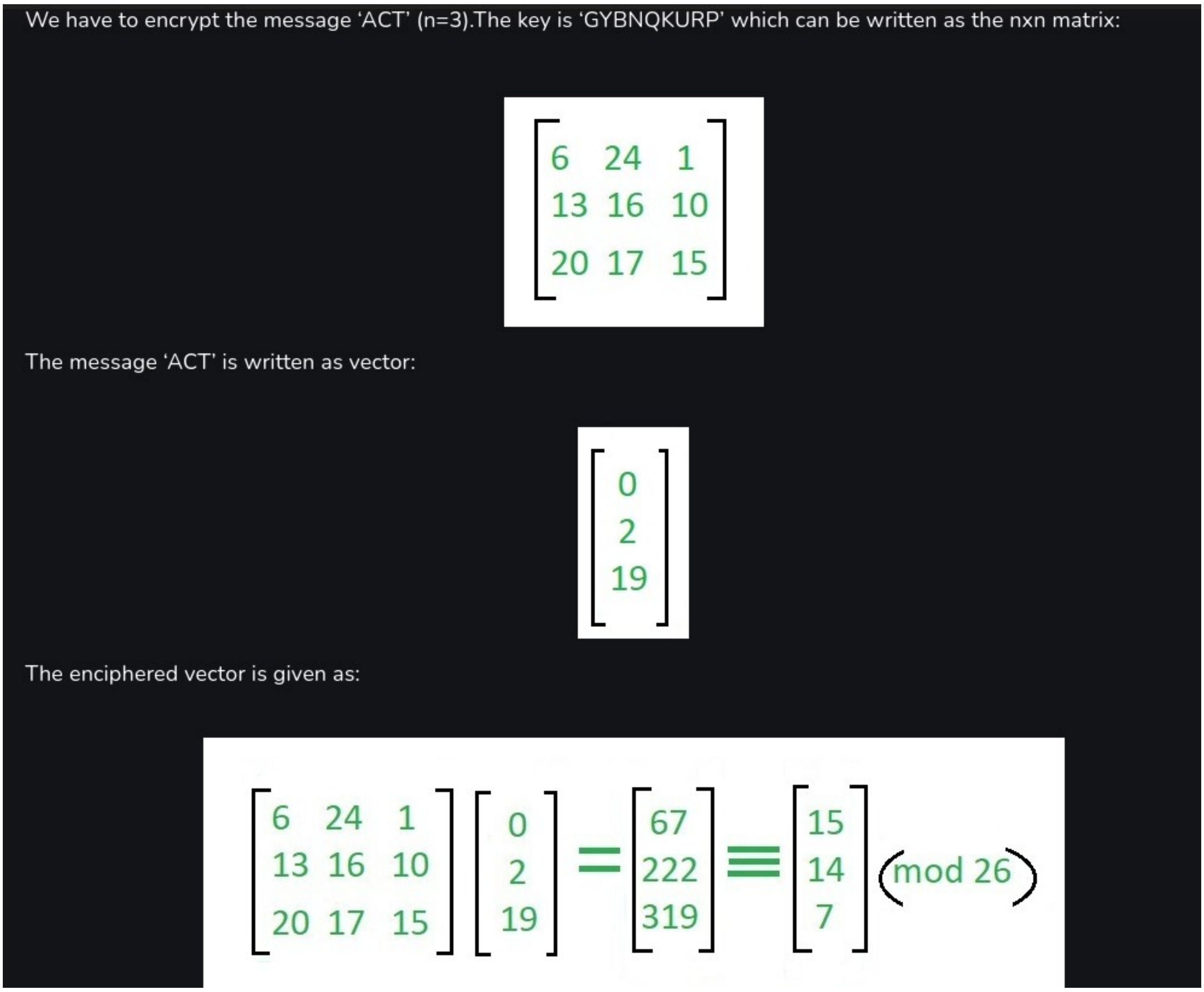


```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 3
Enter text: karan123
Invalid input! Use only alphabets and spaces.
```

Hill Cipher:



```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 5
Enter text: ACT
Enter matrix size (n): 3
Enter key (9 letters): GYBNQKURP
Encrypted: POH
```



We have to encrypt the message ‘ACT’ (n=3).The key is ‘GYBNQKURP’ which can be written as the nxn matrix:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The message ‘ACT’ is written as vector:

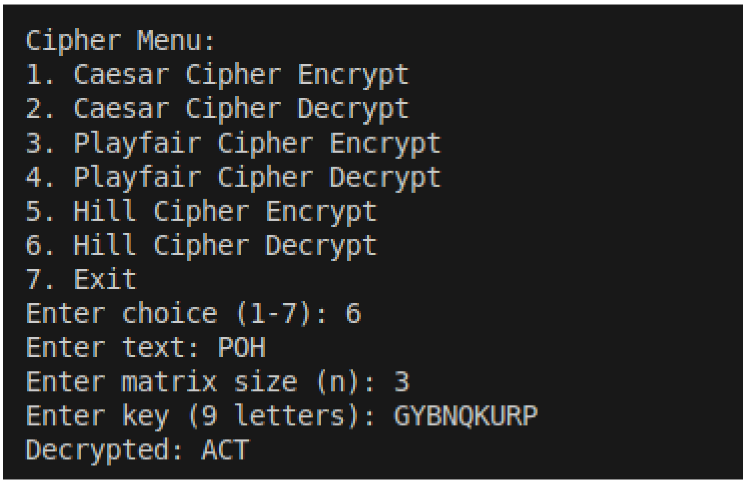
$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

The enciphered vector is given as:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \pmod{26}$$

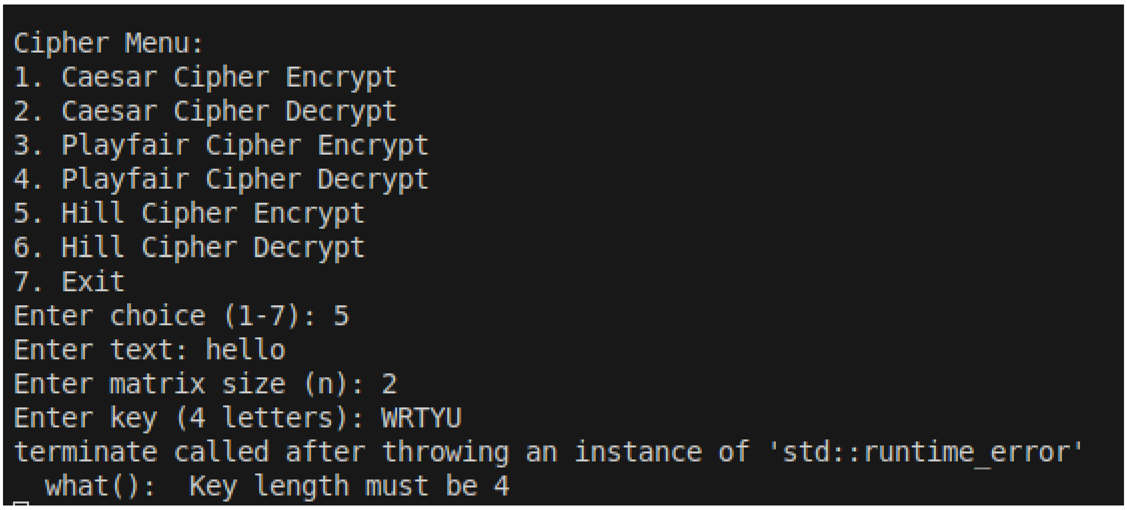
which corresponds to cipher text of ‘POH’.

Decryption case of above:



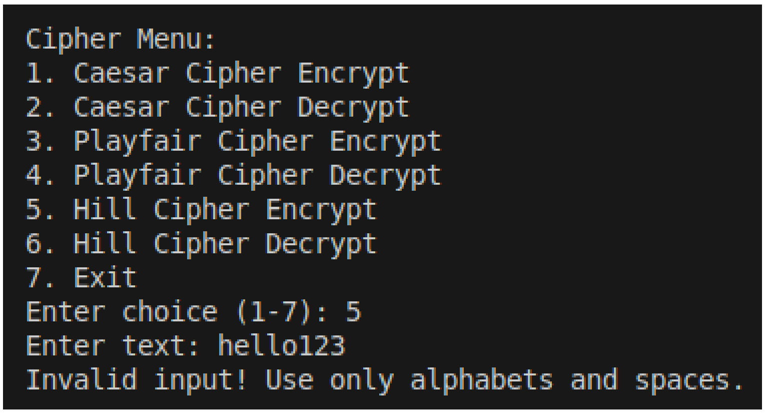
```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 6
Enter text: POH
Enter matrix size (n): 3
Enter key (9 letters): GYBNQKURP
Decrypted: ACT
```

Wrong key size:



```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 5
Enter text: hello
Enter matrix size (n): 2
Enter key (4 letters): WRTYU
terminate called after throwing an instance of 'std::runtime_error'
  what():  Key length must be 4
```

Wrong Input:



```
Cipher Menu:
1. Caesar Cipher Encrypt
2. Caesar Cipher Decrypt
3. Playfair Cipher Encrypt
4. Playfair Cipher Decrypt
5. Hill Cipher Encrypt
6. Hill Cipher Decrypt
7. Exit
Enter choice (1-7): 5
Enter text: hello123
Invalid input! Use only alphabets and spaces.
```