Suraj Raju Lokapure | 2020BTECS00081 | B5 batch

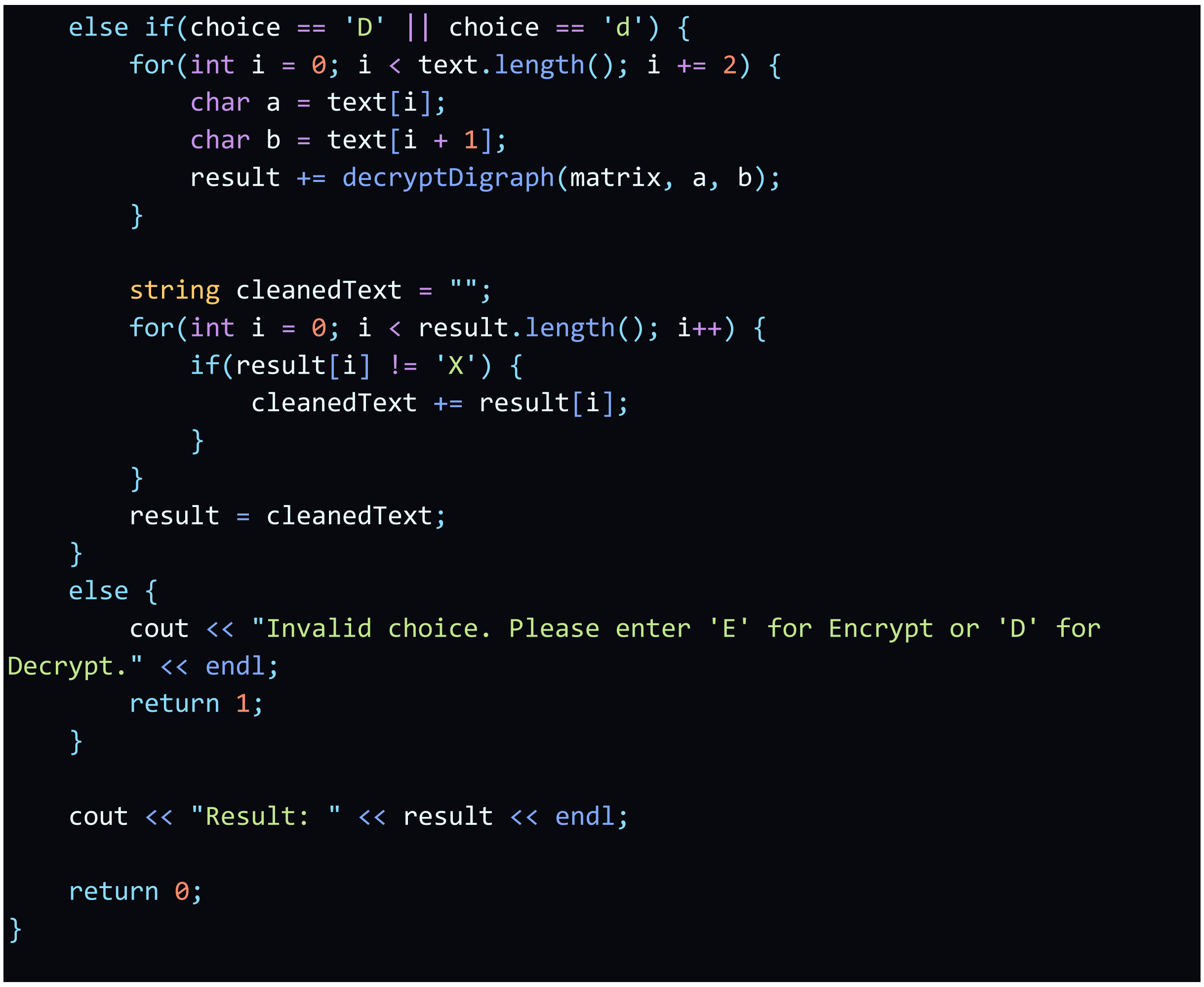
CNS lab | Experiment 03

Aim: To implement Playfair cipher

Theory: The Playfair Cipher is a digraph substitution cipher that encrypts pairs of letters at a time. It uses a 5x5 grid containing a keyword (omitting duplicates) for encryption. To encrypt, you find each pair of letters in the grid and apply specific rules. If the letters are in the same row, they are replaced with the letters to their right, and if in the same column, with the letters below. If neither, they form a rectangle and are replaced with the letters at the opposite corners.

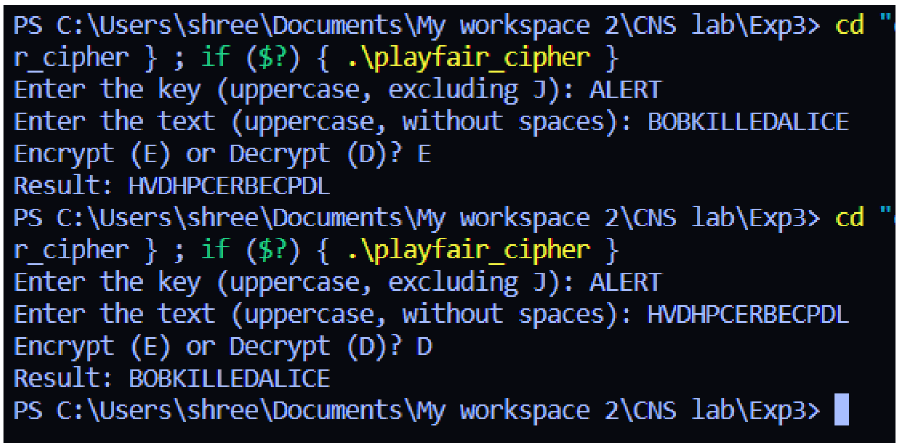
Code:

```cpp
#include <iostream>
#include <string>
using namespace std;

const int SIZE = 5;

void generateMatrix(string key, char matrix[SIZE][SIZE]) {
    string keyWithoutDuplicates = "";
    bool used[26] = { false };

    for(char c : key) {
        if(c != 'J' && !used[c - 'A']) {
            keyWithoutDuplicates += c;
            used[c - 'A'] = true;
        }
    }

    int index = 0;
    for(int i = 0; i < SIZE; i++) {
        for(int j = 0; j < SIZE; j++) {
            if(index < keyWithoutDuplicates.length()) {
                matrix[i][j] = keyWithoutDuplicates[index++];
            } else {
                for(char c = 'A'; c <= 'Z'; c++) {
                    if(c != 'J' && !used[c - 'A']) {
                        matrix[i][j] = c;
                        used[c - 'A'] = true;
                        break;
                    }
                }
            }
        }
    }
}

void findPosition(char matrix[SIZE][SIZE], char c, int& row, int& col) {
    if (c == 'J') c = 'I';

    for(int i = 0; i < SIZE; i++) {
        for(int j = 0; j < SIZE; j++) {
            if(matrix[i][j] == c) {
                row = i;
                col = j;
                return;
            }
        }
    }
}

string encryptDigraph(char matrix[SIZE][SIZE], char a, char b) {
    int rowA, colA, rowB, colB;
    findPosition(matrix, a, rowA, colA);
    findPosition(matrix, b, rowB, colB);

    if(rowA == rowB) {
        return string(1, matrix[rowA][(colA + 1) % SIZE]) + string(1,
matrix[rowB][(colB + 1) % SIZE]);
    }
    if(colA == colB) {
        return string(1, matrix[(rowA + 1) % SIZE][colA]) + string(1,
matrix[(rowB + 1) % SIZE][colB]);
    }
    return string(1, matrix[rowA][colB]) + string(1, matrix[rowB][colA]);
}

string decryptDigraph(char matrix[SIZE][SIZE], char a, char b) {
    int rowA, colA, rowB, colB;
    findPosition(matrix, a, rowA, colA);
    findPosition(matrix, b, rowB, colB);


    if(rowA == rowB) {
        return string(1, matrix[rowA][(colA - 1 + SIZE) % SIZE]) + string(1,
matrix[rowB][(colB - 1 + SIZE) % SIZE]);
    }

    if(colA == colB) {
        return string(1, matrix[(rowA - 1 + SIZE) % SIZE][colA]) + string(1,
matrix[(rowB - 1 + SIZE) % SIZE][colB]);
    }

    return string(1, matrix[rowA][colB]) + string(1, matrix[rowB][colA]);
}

int main() {
    string key, text;
    char matrix[SIZE][SIZE];

    cout << "Enter the key (uppercase, excluding J): ";
    cin >> key;

    generateMatrix(key, matrix);

    cout << "Enter the text (uppercase, without spaces): ";
    cin >> text;

    string result;
    char choice;

    cout << "Encrypt (E) or Decrypt (D)? ";
    cin >> choice;

    if(choice == 'E' || choice == 'e') {

        string preparedText = "";
        for(int i = 0; i < text.length(); i += 2) {
            if(i + 1 < text.length()) {
                if(text[i] == text[i + 1]) {
                    preparedText += text[i];
                    preparedText += 'X';
                    i--;
                } else {
                    preparedText += text.substr(i, 2);
                }
            } else {
                preparedText += text[i];
                preparedText += 'X';
            }
        }
        for(int i = 0; i < preparedText.length(); i += 2) {
            char a = preparedText[i];
            char b = preparedText[i + 1];
            result += encryptDigraph(matrix, a, b);
        }
    }
```



```
    else if(choice == 'D' || choice == 'd') {
        for(int i = 0; i < text.length(); i += 2) {
            char a = text[i];
            char b = text[i + 1];
            result += decryptDigraph(matrix, a, b);
        }

        string cleanedText = "";
        for(int i = 0; i < result.length(); i++) {
            if(result[i] != 'X') {
                cleanedText += result[i];
            }
        }
        result = cleanedText;
    }
    else {
        cout << "Invalid choice. Please enter 'E' for Encrypt or 'D' for
Decrypt." << endl;
        return 1;
    }

    cout << "Result: " << result << endl;

    return 0;
}
```

Output:



```
PS C:\Users\shree\Documents\My workspace 2\CNS lab\Exp3> cd "
r_cipher } ; if ($?) { .\playfair_cipher }
Enter the key (uppercase, excluding J): ALERT
Enter the text (uppercase, without spaces): BOBKILLEDALICE
Encrypt (E) or Decrypt (D)? E
Result: HVDHPCERBECPDL
PS C:\Users\shree\Documents\My workspace 2\CNS lab\Exp3> cd "
r_cipher } ; if ($?) { .\playfair_cipher }
Enter the key (uppercase, excluding J): ALERT
Enter the text (uppercase, without spaces): HVDHPCERBECPDL
Encrypt (E) or Decrypt (D)? D
Result: BOBKILLEDALICE
PS C:\Users\shree\Documents\My workspace 2\CNS lab\Exp3>
```