Ex. No.: 3 **Date:**

PLAYFAIR CIPHER

Problem Statement:

The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets (digraphs) instead of a single alphabet.

.

The Algorithm consists of 2 steps:

I. **Generate the key Square(5×5):**
   - The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by
     I.
   - The initial alphabets in the key square are the unique alphabets of the key in the order in which

II. **Algorithm to encrypt the plain text:** The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

Rules for Encryption:

- **If both the letters are in the same column**: Take the letter below each one (going back to the top if at the bottom).
- **If both the letters are in the same row**: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).
- **If neither of the above rules is true**: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Aim:

To implement Playfair Cipher technique using C.

Algorithm:

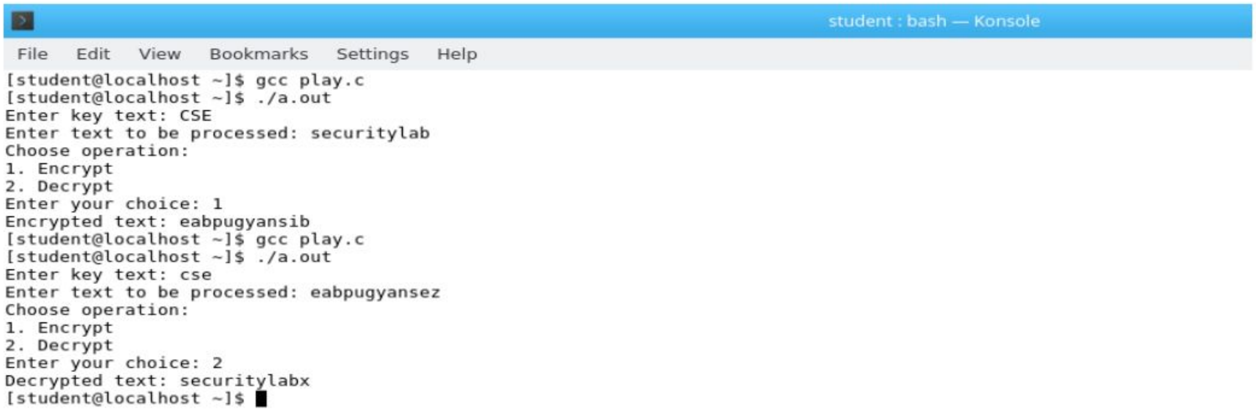
1. Initialize the contents of the table to zero.
2. Get the length of the key
3. Get the key string from the user.
4. Insert each element of the key into the table.
5. Fill the remaining entries of the table with the character not already entered into the table.
6. Enter the length of the plaintext.
7. Get the plaintext string.

Program Code:

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

#define SIZE 30

void toLowerCase(char plain[], int ps) {
   for (int i = 0; i < ps; i++) {
      if (plain[i] > 64 && plain[i] < 91)
         plain[i] += 32;
   }
}

int removeSpaces(char* plain, int ps) {
   int i, count = 0;
   for (i = 0; i < ps; i++)
      if (plain[i] != ' ')
         plain[count++] = plain[i];
   plain[count] = '\0';
   return count;
}

void generateKeyTable(char key[], int ks, char keyT[5][5]) {
   int i, j, k, flag = 0, *dicty;
   dicty = (int*)calloc(26, sizeof(int));

   for (i = 0; i < ks; i++) {
      if (key[i] != 'j')
         dicty[key[i] - 97] = 2;
   }

   dicty['j' - 97] = 1;

   i = 0;
   j = 0;
   for (k = 0; k < ks; k++) {
      if (dicty[key[k] - 97] == 2) {
         dicty[key[k] - 97] -= 1;
         keyT[i][j] = key[k];
         j++;
         if (j == 5) {
            i++;
            j = 0;
         }
      }
   }

   for (k = 0; k < 26; k++) {
      if (dicty[k] == 0) {
         keyT[i][j] = (char)(k + 97);
```

```
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
}

void search(char keyT[5][5], char a, char b, int arr[]) {
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}

int mod5(int a) {
    if (a < 0)
        a += 5;
    return (a % 5);
}

int prepare(char str[], int ptrs) {
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

void encrypt(char str[], char keyT[5][5], int ps) {
    int i, a[4];

    for (i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

void decrypt(char str[], char keyT[5][5], int ps) {
    int i, a[4];
    for (i = 0; i < ps; i += 2) {
        search(keyT, str[i], str[i + 1], a);
        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] - 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] - 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

void encryptAndDecryptByPlayfairCipher(char str[], char key[], int choice) {
    char ps, ks, keyT[5][5];

    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    if (choice == 1) {
        encrypt(str, keyT, ps);
        printf("Encrypted text: %s\n", str);
    }
    else if (choice == 2) {
        decrypt(str, keyT, ps);
        printf("Decrypted text: %s\n", str);
    }
}

int main() {
    char str[SIZE], key[SIZE];
    int choice;

    printf("Enter key text: ");
    scanf("%s", key);

    printf("Enter text to be processed: ");
    scanf("%s", str);

    printf("Choose operation:\n");
    printf("1. Encrypt\n2. Decrypt\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    encryptAndDecryptByPlayfairCipher(str, key, choice);

    return 0;
}
```

Output:



```
[student@localhost ~]$ gcc play.c
[student@localhost ~]$ ./a.out
Enter key text: CSE
Enter text to be processed: securitylab
Choose operation:
1. Encrypt
2. Decrypt
Enter your choice: 1
Encrypted text: eabpugyansib
[student@localhost ~]$ gcc play.c
[student@localhost ~]$ ./a.out
Enter key text: cse
Enter text to be processed: eabpugyansez
Choose operation:
1. Encrypt
2. Decrypt
Enter your choice: 2
Decrypted text: securitylabx
[student@localhost ~]$ █
```

Result: