Name: Syed Irfan Faraz Id:21BCT0170

**Assessment -1**

**Q) Implement the following substitution cipher techniques without using standard**

**cryptographic library Functions.**

**i. Playfair Cipher**

**ii. Hill cipher**

**iii. Auto key pair Cipher**

i)

**Aim: Play Fair Cipher**

**Code**:  
  
#include <bits/stdc++.h>

#include <cstring>

using namespace std;

class PFEncryption {

private:

char alphabets[5][5];

char uniqueChar[26];

string ch = “ABCDEFGHIKLMNOPQRSTUVWXYZ”;

string encrypted = “”;

string decrypted = “”;

public:

void makeArray(string keyword) {

keyword = toUpper(keyword);

replaceJWithI(keyword);

bool present;

int val = 0;

int uniqueLen;

for (int I = 0; I < keyword.length(); i++) {

present = false;

uniqueLen = 0;

if (keyword[i] != ‘ ‘) {

for (int k = 0; k < 26; k++) {

if (uniqueChar[k] == ‘\0’) {

break;

}

uniqueLen++;

}

for (int j = 0; j < uniqueLen; j++) {

if (keyword[i] == uniqueChar[j]) {

present = true;

}

}

if (!present) {

uniqueChar[val] = keyword[i];

val++;

}

}

ch.erase(remove(ch.begin(), ch.end(), keyword[i]), ch.end());

}

for (int I = 0; I < ch.length(); i++) {

uniqueChar[val] = ch[i];

val++;

}

val = 0;

for (int I = 0; I < 5; i++) {

for (int j = 0; j < 5; j++) {

alphabets[i][j] = uniqueChar[val];

val++;

cout << alphabets[i][j] << “\t”;

}

cout << endl;

}

}

string manageMessage(string msg) {

int val = 0;

int len = msg.length() – 2;

string newTxt = “”;

string intermediate = “”;

while (len >= 0) {

intermediate = msg.substr(val, 2);

if (intermediate[0] == intermediate[1]) {

newTxt = intermediate[0] + “x” + intermediate[1];

msg.replace(msg.find(intermediate), 2, newTxt);

len++;

}

len -= 2;

val += 2;

}

if (msg.length() % 2 != 0) {

msg += ‘x’;

}

return toUpper(replaceJWithI(removeSpaces(msg)));

}

void doPlayFair(string msg, string tag) {

int val = 0;

while (val < msg.length()) {

searchAndEncryptOrDecrypt(msg.substr(val, 2), tag);

val += 2;

}

}

void searchAndEncryptOrDecrypt(string doubblyCh, string tag) {

char ch1 = doubblyCh[0];

char ch2 = doubblyCh[1];

int row1 = 0, col1 = 0, row2 = 0, col2 = 0;

for (int I = 0; I < 5; i++) {

for (int j = 0; j < 5; j++) {

if (alphabets[i][j] == ch1) {

row1 = I;

col1 = j;

} else if (alphabets[i][j] == ch2) {

row2 = I;

col2 = j;

}

}

}

if (tag == “Encrypt”)

encrypt(row1, col1, row2, col2);

else if (tag == “Decrypt”)

decrypt(row1, col1, row2, col2);

}

void encrypt(int row1, int col1, int row2, int col2) {

if (row1 == row2) {

col1 = col1 + 1;

col2 = col2 + 1;

if (col1 > 4)

col1 = 0;

if (col2 > 4)

col2 = 0;

encrypted += (alphabets[row1][col1]);

encrypted += (alphabets[row1][col2]);

} else if (col1 == col2) {

row1 = row1 + 1;

row2 = row2 + 1;

if (row1 > 4)

row1 = 0;

if (row2 > 4)

row2 = 0;

encrypted += (alphabets[row1][col1]);

encrypted += (alphabets[row2][col1]);

} else {

encrypted += (alphabets[row1][col2]);

encrypted += (alphabets[row2][col1]);

}

}

void decrypt(int row1, int col1, int row2, int col2) {

if (row1 == row2) {

col1 = col1 – 1;

col2 = col2 – 1;

if (col1 < 0)

col1 = 4;

if (col2 < 0)

col2 = 4;

decrypted += (alphabets[row1][col1]);

decrypted += (alphabets[row1][col2]);

} else if (col1 == col2) {

row1 = row1 – 1;

row2 = row2 – 1;

if (row1 < 0)

row1 = 4;

if (row2 < 0)

row2 = 4;

decrypted += (alphabets[row1][col1]);

decrypted += (alphabets[row2][col1]);

} else {

decrypted += (alphabets[row1][col2]);

decrypted += (alphabets[row2][col1]);

}

}

string getEncrypted() {

return encrypted;

}

string getDecrypted() {

return decrypted;

}

private:

string toUpper(string str) {

for (auto &c : str) {

c = toupperI;

}

return str;

}

string replaceJWithI(string str) {

for (auto &c : str) {

if (c == ‘J’) {

c = ‘I’;

}

}

return str;

}

string removeSpaces(string str) {

str.erase(remove(str.begin(), str.end(), ‘ ‘), str.end());

return str;

}

};

int main() {

string key, msg;

cout << “Enter keyword: “;

getline(cin, key);

cout << “Enter message to encrypt: “;

getline(cin, msg);

PFEncryption pfEncryption;

pfEncryption.makeArray(key);

msg = pfEncryption.manageMessage(msg);

pfEncryption.doPlayFair(msg, “Encrypt”);

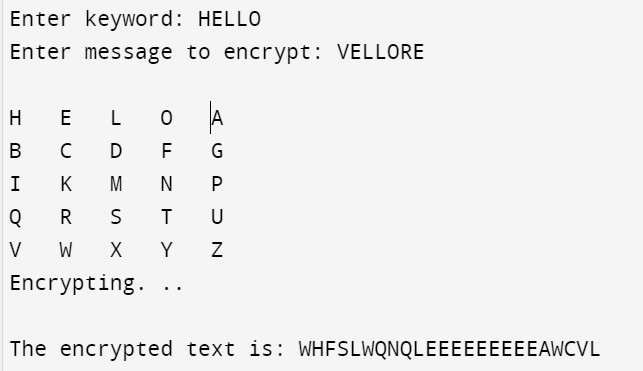
string en = pfEncryption.getEncrypted();

cout << “Encrypting. .. \n\nThe encrypted text is: “ << en << endl;

return 0;

}

**Input and Output:**



ii) **Aim: Hill Cipher**

**Code:**

#include <bits/stdc++.h>

#include <string>

using namespace std;

class HillCipher {

public:

static int keymat[3][3];

static int invkeymat[3][3];

static string key;

static string encrypt(char a, char b, char c) {

string ret = "";

int x, y, z;

int posa = (int)a - 65;

int posb = (int)b - 65;

int posc = (int)c - 65;

x = posa \* keymat[0][0] + posb \* keymat[1][0] + posc \* keymat[2][0];

y = posa \* keymat[0][1] + posb \* keymat[1][1] + posc \* keymat[2][1];

z = posa \* keymat[0][2] + posb \* keymat[1][2] + posc \* keymat[2][2];

a = key[(x % 26 + 26) % 26];

b = key[(y % 26 + 26) % 26];

c = key[(z % 26 + 26) % 26];

ret = string(1, a) + string(1, b) + string(1, c);

return ret;

}

static string decrypt(char a, char b, char c) {

string ret = "";

int x, y, z;

int posa = (int)a - 65;

int posb = (int)b - 65;

int posc = (int)c - 65;

x = posa \* invkeymat[0][0] + posb \* invkeymat[1][0] + posc \* invkeymat[2][0];

y = posa \* invkeymat[0][1] + posb \* invkeymat[1][1] + posc \* invkeymat[2][1];

z = posa \* invkeymat[0][2] + posb \* invkeymat[1][2] + posc \* invkeymat[2][2];

a = key[(x % 26 + 26) % 26];

b = key[(y % 26 + 26) % 26];

c = key[(z % 26 + 26) % 26];

ret = string(1, a) + string(1, b) + string(1, c);

return ret;

}

};

int HillCipher::keymat[3][3] = {{1, 2, 1}, {2, 3, 2}, {2, 2, 1}};

int HillCipher::invkeymat[3][3] = {{-1, 0, 1}, {2, -1, 0}, {-2, 2, -1}};

string HillCipher::key = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

int main() {

string text, outtext = "", outtext1 = "";

int n;

cout << "Enter the Plain text for Encryption: ";

getline(cin, text);

text = text.substr(0, text.find("\n"));

text = text.substr(0, text.find("\r"));

text = text.substr(0, text.find(" "));

//text = toUpper(text);

text.erase(remove(text.begin(), text.end(), ' '), text.end()); // removing spaces

n = text.length() % 3;

if (n != 0) {

for (int i = 1; i <= (3 - n); i++) {

text += 'X';

}

}

cout << "Padded Text: " << text << endl;

char ptextchars[text.length() + 1];

strcpy(ptextchars, text.c\_str());

for (int i = 0; i < text.length(); i += 3) {

outtext += HillCipher::encrypt(ptextchars[i], ptextchars[i + 1], ptextchars[i + 2]);

}

cout << "Encrypted Message: " << outtext << endl;

char ptextchars1[outtext.length() + 1];

strcpy(ptextchars1, outtext.c\_str());

for (int i = 0; i < outtext.length(); i += 3) {

outtext1 += HillCipher::decrypt(ptextchars1[i], ptextchars1[i + 1], ptextchars1[i + 2]);

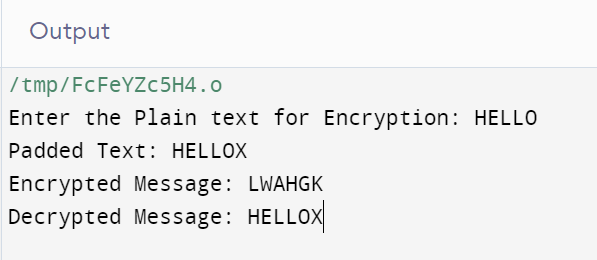
}

cout << "Decrypted Message: " << outtext1 << endl;

return 0;

}

***Sample Input and Output:***



iii)

**Aim: Auto Key Pair Cipher**

**Code:**

#include<iostream>

#include<vector>

using namespace std;

#define DB(x) cout<<#x<<": "<<x<<'\n'

class AutoKeySystem {

const string AlphabetSet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string key;

public:

string encryption(string plainText,string key){

int m = (int)key.size() , n = (int)plainText.size();

string cipherText = "";

for(int i = 0; i < m ; i++){

cipherText += AlphabetSet[( (plainText[i] - 65) + (key[i] - 65) ) %26];

}

int k = 0;

for(int i = m;i<n; i++){

cipherText += AlphabetSet[( (plainText[i] - 65) + (plainText[k % n] - 65) ) %26];

k++;

}

return cipherText;

}

string decryption(const string cipherText,string fullKey){

int n = (int)cipherText.size();

string plainText = "";

int k;

for(int i = 0; i < n ; i++){

k = ( (cipherText[i] - 65) - (fullKey[i] - 65) ) %26;

if(k < 0) k += 26;

plainText += AlphabetSet[k];

}

return plainText;

}

};

int main(){

string plainText,key = "DE";

cout<<"Enter plainText: ";

cin>>plainText;

AutoKeySystem HC;

string cipherText = HC.encryption(plainText,key);

cout<<"Ciphertext: "<<cipherText<<'\n';

string fullkey = key;

int index = 0, PTlength = (int)plainText.size();

for(int i= (int)key.size(); i<cipherText.size() ; i++){

fullkey += plainText[index % PTlength];

index ++;

}

string decryptedText = HC.decryption(cipherText, fullkey);

cout<<"decryptedText: "<<decryptedText<<'\n';

}

**Input and Output:**

