**Practical File**

**Information Security**



B.Sc. (H) Computer Science

Semester 6th

Name: Kartikey Sharma

Roll No: CSC/21/18

Examination Roll No: 21059570020

1. Implement the error correcting code.

#include<iostream>

using namespace std;

int main() {

int d[10];

int dt[10],c,c1,c2,c3,i;

cout<<"Enter 4 bits of data one by one\n";

cin>>d[0];

cin>>d[1];

cin>>d[2];

cin>>d[4];

//Calculation of even parity

d[6]=d[0]^d[2]^d[4];

d[5]=d[0]^d[1]^d[4];

d[3]=d[0]^d[1]^d[2];

cout<<"\nEncoded data is\n";

for(i=0;i<7;i++)

cout<<d[i];

cout<<"\n\nEnter received data bits one by one\n";

for(i=0;i<7;i++)

cin>>dt[i];

c1=dt[6]^dt[4]^dt[2]^dt[0];

c2=dt[5]^dt[4]^dt[1]^dt[0];

c3=dt[3]^dt[2]^dt[1]^dt[0];

c=c3\*4+c2\*2+c1 ;

if(c==0) {

cout<<"\nNo error while transmission of data\n";

}

else {

cout<<"\nError on position "<<c;

cout<<"\nData sent : ";

for(i=0;i<7;i++)

cout<<d[i];

cout<<"\nData received : ";

for(i=0;i<7;i++)

cout<<dt[i];

cout<<"\nCorrect message is\n";

if(dt[7-c]==0)

dt[7-c]=1;

else

dt[7-c]=0;

for (i=0;i<7;i++) {

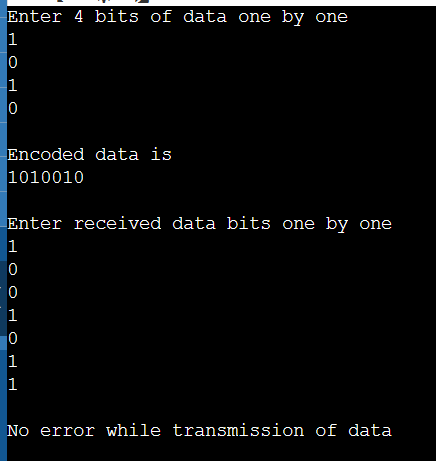
cout<<dt[i];

}

}

return 0;

}



2. Implement the error detecting code.

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

class hamming{

public:

string data; //it is the raw data received

int m , r = 0; // n is the length of raw data and r is the number of redundant bits

char \* msg; // it will store the all bits (data + redundant). We made it dynamic because at compile time we dont know how much redundant bits will be there, we will initialize memory to it once we know the number of redundant bits.

hamming(string data){

this->data = data;

//reversing the data received

reverse(data.begin(),data.end());

m = data.size();

int power = 1;

//finding the number of redundant bits and storing them in r

while(power < (m + r + 1)){

r++;

power\*=2;

}

//Allocating memory to our dynamic msg array(Note we are using one based indexing).

msg = new char[m+r+1];

int curr = 0;

//initializing the msg with data bits and for redundant bits, an initial value of n

for(int i = 1 ; i <= m+r ; i++){

if(i & (i-1)){

msg[i] = data[curr++];

}

else msg[i] = 'n';

}

//function call to set the redundant bits

setRedundantBits();

}

//function to show the whole msg

void showmsg(){

cout << "the data packet to be sent is : ";

for(int i = m+r ; i >= 1 ; i--){

cout << msg[i] << " ";

}

cout << endl;

}

void setRedundantBits(){

int bit = 0;

for(int i = 1 ; i <= m+r ; i\*=2){

int count = 0;

//inner loop runs for data bits

for(int j = i+1 ; j<=m+r ; j++){

if(j & (1 << bit)){

if(msg[j] == '1') count++;

}

}

if(count & 1) msg[i] = '1';

else msg[i] = '0';

//increasing the bit position.

bit++;

}

//showing up the message to be sent(data + redundant)

showmsg();

}

void receiver(){

string ans = "";

int bit = 0;

//this loop corresponds to the logic used in set redundant bits function

for(int i = 1 ; i <= m+r ; i\*=2){

int count = 0;

for(int j = i+1 ; j<=m+r ; j++){

if(j & (1 << bit)){

if(msg[j] == '1') count++;

}

}

//incrementing the ans variable with the parity of redundant bit

// if it was right then add 0 else 1

if(count & 1){

if(msg[i] == '1') ans.push\_back('0');

else ans.push\_back('1');

}

else{

if(msg[i]=='0') ans.push\_back('0');

else ans.push\_back('1');

}

bit++;

}

// if the ans had any occurrence of 1 then there is some fault

if(ans.find('1') != string::npos){

int power = 1;

int wrongbit = 0;

//evaluating the binary expression of ans variable

for(int i = 0 ; i < ans.size() ; i++){

if(ans[i]=='1') wrongbit+=power;

power\*=2;

}

cout << "bit number " << wrongbit << " is wrong and having error " << endl;

}

// if the ans dont have any occurrence of 1 then it is correct

else{

cout << "correct data packet received " << endl;

}

}

};

int main(){

string data = "1011001";

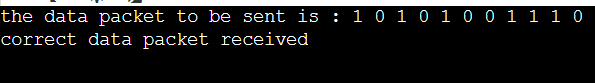
hamming h(data);

//h.msg[i] == '0' ? h.msg[i] = '1' : h.msg[i] = '0';

h.receiver();

return 0;

}



3. Implement caeser cipher substitution operation.

#include<iostream>

#include <string>

using namespace std;

string caesarCipher(const string& message, int shift) {

string result = "";

for (char ch : message) {

if (isalpha(ch)) {

char base = islower(ch) ? 'a' : 'A';

result += static\_cast<char>((ch - base + shift) % 26 + base);

} else {

result += ch;

}

}

return result;

}

int main() {

string message = "kartik";

int shift = 3;

cout << "Original message: " << message << endl;

string encrypted = caesarCipher(message, shift);

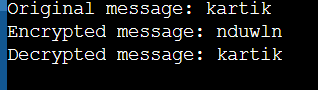
cout << "Encrypted message: " << encrypted << endl;

string decrypted = caesarCipher(encrypted, -shift);

cout << "Decrypted message: " << decrypted << endl;

return 0;

}



4. Implement monoalphabetic and polyalphabetic cipher substitution operation.

#include <iostream>

#include <string>

#include <cctype>

using namespace std;

string monoalphabetic\_encrypt(const string& plaintext, const string& key) {

string ciphertext;

for (char c : plaintext) {

if (isalpha(c)) {

char encrypted\_char = key[toupper(c) - 'A'];

if (islower(c)) {

encrypted\_char = tolower(encrypted\_char);

}

ciphertext += encrypted\_char;

} else {

ciphertext += c;

}

}

return ciphertext;

}

string monoalphabetic\_decrypt(const string& ciphertext, const string& key) {

string plaintext;

for (char c : ciphertext) {

if (isalpha(c)) {

char decrypted\_char = 'A' + key.find(toupper(c));

if (islower(c)) {

decrypted\_char = tolower(decrypted\_char);

}

plaintext += decrypted\_char;

} else {

plaintext += c;

}

}

return plaintext;

}

int main() {

string key = "ZYXWVUTSRQPONMLKJIHGFEDCBA";

string plaintext = "kartikey";

string ciphertext = monoalphabetic\_encrypt(plaintext, key);

cout << "Encrypted Text: " << ciphertext << endl;

string decryptedtext = monoalphabetic\_decrypt(ciphertext, key);

cout << "Decrypted Text: " << decryptedtext << endl;

return 0;

}

#include <iostream>

#include <string>

#include <cctype>

using namespace std;

string vigenere\_encrypt(const string& plaintext, const string& key) {

string ciphertext;

int key\_length = key.length();

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

char c = plaintext[i];

if (isalpha(c)) {

char encrypted\_char = (toupper(c) + toupper(key[i % key\_length]) - 2

\* 'A') % 26 + 'A';

if (islower(c)) {

encrypted\_char = tolower(encrypted\_char);

}

ciphertext += encrypted\_char;

} else {

ciphertext += c;

}

}

return ciphertext;

}

string vigenere\_decrypt(const string& ciphertext, const string& key) {

string plaintext;

int key\_length = key.length();

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

char c = ciphertext[i];

if (isalpha(c)) {

char decrypted\_char = (toupper(c) - toupper(key[i % key\_length]) +

26) % 26 + 'A';

if (islower(c)) {

decrypted\_char = tolower(decrypted\_char);

}

plaintext += decrypted\_char;

} else {

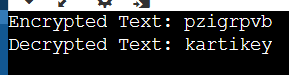
plaintext += c;

}

}

return plaintext;

}



5. Implement playfair cipher substitution operation.

#include <iostream>

#include <string>

#include <cctype>

#include <algorithm>

using namespace std;

// Function to generate the Playfair cipher grid

void generateGrid(const string& key, char grid[5][5]) {

string tempKey = key;

string alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ";

// Remove non-alpha characters and convert to upper case

tempKey.erase(remove\_if(tempKey.begin(), tempKey.end(), [](char c) {

return !isalpha(c);

}), tempKey.end());

// Convert to upper case

transform(tempKey.begin(), tempKey.end(), tempKey.begin(), ::toupper);

tempKey.erase(remove(tempKey.begin(), tempKey.end(), 'J'), tempKey.end());

tempKey.erase(unique(tempKey.begin(), tempKey.end()), tempKey.end());

int k = 0;

for (char c : tempKey) {

grid[k / 5][k % 5] = c;

++k;

}

for (char c : alphabet) {

if (c != 'J' && tempKey.find(c) == string::npos) {

grid[k / 5][k % 5] = c;

++k;

}

}

}

string encrypt(const string& plaintext, const string& key) {

char grid[5][5];

generateGrid(key, grid);

string encryptedText;

for (size\_t i = 0; i < plaintext.length(); i += 2) {

char a = toupper(plaintext[i]);

char b = (i + 1 < plaintext.length()) ? toupper(plaintext[i + 1]) : 'X'; // Append 'X' if the last

int rowA, colA, rowB, colB;

// Find positions of characters in the grid

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

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

if (grid[row][col] == a) {

rowA = row;

colA = col;

}

if (grid[row][col] == b) {

rowB = row;

colB = col;

}

}

}

if (rowA == rowB) {

encryptedText += grid[rowA][(colA + 1) % 5];

encryptedText += grid[rowB][(colB + 1) % 5];

}

else if (colA == colB) {

encryptedText += grid[(rowA + 1) % 5][colA];

encryptedText += grid[(rowB + 1) % 5][colB];

}

else {

encryptedText += grid[rowA][colB];

encryptedText += grid[rowB][colA];

}

}

return encryptedText;

}

int main() {

string key = " string key ";

string plaintext = " this is the plain text ";

string encryptedText = encrypt(plaintext, key);

cout << "Encrypted Text: " << encryptedText << endl;

return 0;

}



6. Implement hill cipher substitution operation.

#include <iostream>

#include <vector>

using namespace std;

// Function to get the key matrix for the Hill cipher

void getKeyMatrix(string key, vector<vector<int>>& keyMatrix) {

int k = 0;

for (int i = 0; i < keyMatrix.size(); i++) {

for (int j = 0; j < keyMatrix[i].size(); j++) {

keyMatrix[i][j] = (key[k]) % 65;

k++;

}

}

}

// Function to encrypt the message using the key matrix

void encrypt(vector<vector<int>>& cipherMatrix, vector<vector<int>>& keyMatrix, vector<vector<int>>& messageVector) {

int x, i, j;

for (i = 0; i < keyMatrix.size(); i++) {

for (j = 0; j < 1; j++) {

cipherMatrix[i][j] = 0;

for (x = 0; x < keyMatrix.size(); x++) {

cipherMatrix[i][j] += keyMatrix[i][x] \* messageVector[x][j];

}

cipherMatrix[i][j] = cipherMatrix[i][j] % 26;

}

}

}

// Hill Cipher encryption

void HillCipher(string message, string key) {

// Assuming keyMatrix is 3x3

vector<vector<int>> keyMatrix(3, vector<int>(3));

getKeyMatrix(key, keyMatrix);

// Creating message vector

vector<vector<int>> messageVector(3, vector<int>(1));

for (int i = 0; i < 3; i++)

messageVector[i][0] = (message[i]) % 65;

// Creating cipher matrix

vector<vector<int>> cipherMatrix(3, vector<int>(1));

// Encrypting the message

encrypt(cipherMatrix, keyMatrix, messageVector);

// Generating the encrypted text

string CipherText;

for (int i = 0; i < 3; i++)

CipherText += cipherMatrix[i][0] + 65;

cout << "Ciphertext: " << CipherText << endl;

}

int main() {

string message = "ACT"; // Your message here

string key = "GYBNQKURP"; // Your key here

HillCipher(message, key);

return 0;

}



Practical Ques. P7

7. Implement rail fence cipher transposition operation.

// C++ program to illustrate Rail Fence Cipher

// Encryption and Decryption

#include <bits/stdc++.h>

using namespace std;

// function to encrypt a message

string encryptRailFence(string text, int key)

{

// create the matrix to cipher plain text

// key = rows , length(text) = columns

char rail[key][(text.length())];

// filling the rail matrix to distinguish filled

// spaces from blank ones

for (int i=0; i < key; i++)

for (int j = 0; j < text.length(); j++)

rail[i][j] = '\n';

// to find the direction

bool dir\_down = false;

int row = 0, col = 0;

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

{

// check the direction of flow

// reverse the direction if we've just

// filled the top or bottom rail

if (row == 0 || row == key-1)

dir\_down = !dir\_down;

// fill the corresponding alphabet

rail[row][col++] = text[i];

// find the next row using direction flag

dir\_down?row++ : row--;

}

//now we can construct the cipher using the rail matrix

string result;

for (int i=0; i < key; i++)

for (int j=0; j < text.length(); j++)

if (rail[i][j]!='\n')

result.push\_back(rail[i][j]);

return result;

}

// This function receives cipher-text and key

// and returns the original text after decryption

string decryptRailFence(string cipher, int key)

{

// create the matrix to cipher plain text

// key = rows , length(text) = columns

char rail[key][cipher.length()];

// filling the rail matrix to distinguish filled

// spaces from blank ones

for (int i=0; i < key; i++)

for (int j=0; j < cipher.length(); j++)

rail[i][j] = '\n';

// to find the direction

bool dir\_down;

int row = 0, col = 0;

// mark the places with '\*'

for (int i=0; i < cipher.length(); i++)

{

// check the direction of flow

if (row == 0)

dir\_down = true;

if (row == key-1)

dir\_down = false;

// place the marker

rail[row][col++] = '\*';

// find the next row using direction flag

dir\_down?row++ : row--;

}

// now we can construct the fill the rail matrix

int index = 0;

for (int i=0; i<key; i++)

for (int j=0; j<cipher.length(); j++)

if (rail[i][j] == '\*' && index<cipher.length())

rail[i][j] = cipher[index++];

// now read the matrix in zig-zag manner to construct

// the resultant text

string result;

row = 0, col = 0;

for (int i=0; i< cipher.length(); i++)

{

// check the direction of flow

if (row == 0)

dir\_down = true;

if (row == key-1)

dir\_down = false;

// place the marker

if (rail[row][col] != '\*')

result.push\_back(rail[row][col++]);

// find the next row using direction flag

dir\_down?row++: row--;

}

return result;

}

//driver program to check the above functions

int main()

{

cout << encryptRailFence("attack at once", 2) << endl;

cout << encryptRailFence("defend the east wall", 3) << endl;

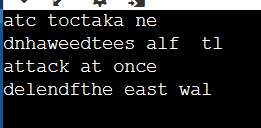
//Now decryption of the same cipher-text

cout << decryptRailFence("atc toctaka ne",2) << endl;

cout << decryptRailFence("dnhaweedtees alf tl",3) << endl;

return 0;

}



Q8) Implement row transposition cipher transposition operation.

#include <iostream>

#include <string>

#include <algorithm>

#include <vector>

std::string encrypt(const std::string &text, const std::vector<int> &key) {

int columns = key.size();

int rows = text.length() / columns + (text.length() % columns ? 1 : 0);

std::vector<std::vector<char>> grid(rows, std::vector<char>(columns, '\*'));

// Fill the grid with the plaintext

for (int i = 0, k = 0; i < rows; ++i) {

for (int j = 0; j < columns && k < text.length(); ++j) {

grid[i][j] = text[k++];

}

}

// Read the grid columns by the order of the key to get the ciphertext

std::string ciphertext;

for (int k : key) {

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

if (grid[i][k - 1] != '\*') {

ciphertext += grid[i][k - 1];

}

}

}

return ciphertext;

}

int main() {

std::string text = "This is a secret message";

std::vector<int> key = {3, 1, 4, 2}; // Example key

// Remove spaces and convert to uppercase

text.erase(remove(text.begin(), text.end(), ' '), text.end());

std::transform(text.begin(), text.end(), text.begin(), ::toupper);

std::string encrypted = encrypt(text, key);

std::cout << "Encrypted: " << encrypted << std::endl;

return 0;

}



9. Implement product cipher transposition operation.

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

// Function to perform the transposition operation

std::string transpositionCipher(const std::string& plaintext, const std::string& key) {

std::string ciphertext;

int keyLength = key.length();

// Calculate the number of rows required for transposition

int numRows = plaintext.length() / keyLength;

if (plaintext.length() % keyLength != 0)

numRows++;

// Create a 2D matrix to store the characters

std::vector<std::vector<char>> matrix(numRows, std::vector<char>(keyLength, ' '));

// Fill the matrix with the plaintext characters row by row

int index = 0;

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

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

if (index < plaintext.length()) {

matrix[i][j] = plaintext[index];

index++;

}

}

}

// Create a permutation of the key to determine the order of columns

std::string sortedKey = key;

std::sort(sortedKey.begin(), sortedKey.end());

// Read characters from the matrix column by column using the sorted key

for (char ch : sortedKey) {

int col = key.find(ch);

for (int row = 0; row < numRows; ++row) {

ciphertext += matrix[row][col];

}

}

return ciphertext;

}

int main() {

std::string plaintext = "KARTIKEY";

std::string key = "KEY";

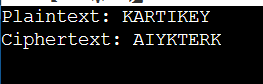
std::string ciphertext = transpositionCipher(plaintext, key);

std::cout << "Plaintext: " << plaintext << std::endl;

std::cout << "Ciphertext: " << ciphertext << std::endl;

return 0;

}



Q11) Implement a stream cipher technique.

#include <iostream>

#include <string>

std::string encryptDecrypt(const std::string& message, const std::string& key) {

std::string result;

int keyLength = key.length();

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

result += message[i] ^ key[i % keyLength]; // XOR operation

}

return result;

}

int main() {

std::string plaintext = "KARTIKEY";

std::string key = "SECRET";

std::string ciphertext = encryptDecrypt(plaintext, key);

std::cout << "Ciphertext: " << ciphertext << std::endl;

std::string decryptedText = encryptDecrypt(ciphertext, key);

std::cout << "Decrypted Text: " << decryptedText << std::endl;

return 0;

}

