CEN 593 – Computer Networks
Laboratory File

BTech Computer Engineering
Vth Semester

Submitted by:
Abdul Basit (20BCS004)

Submitted to:
Professor M Amjad
Mr. Hannan Mansoor

Department of Computer Engineering,
Faculty of Engineering & Technology,
Jamia Millia Islamia, New Delhi

2022

Sr No	Name of Program	Page No	Date
1.	Caesar Cipher	3-6	20/07/22
2.	Transposition Cipher	7-11	27/07/22
3.	Baconian Cipher	12-14	17/08/22
4.	Server-Client Socket Program	15-16	31/08/22
5.	Server-Client with Substitution Cipher	17-19	31/08/22
6.	Client-Server-Multiple Client Broadcasting	20-23	07/09/22
7.	Check the Datatype of Input from the Client on Server	24-25	12/10/22
8.	Server-Client with Rail-Fencing Cipher	26-30	19/10/22
9.	Server-Client with Vigenère Cipher	31-33	12/11/22
10.	**Play-Fair Cipher**	34-40	**19/11/22**

Program 1 :

Caesar cipher

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

int main()
{
    cout << "20BCS004_Abdul Basit" << endl;

    int choice, key;
    char input[1000];
    int temp1[1000], temp2[1000];
    cout << "Enter the Input text: ";
    gets(input);

    cout << "\n";

    char diction[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'w', 'x', 'y', 'z'};
    char dictionCap[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
'V', 'W', 'X', 'Y', 'Z'};

    cout << "Enter the key: ";
    cin >> key;

    for (int i = 0; input[i] != '\0'; i++)
    {
        for (int j = 0; diction[j] != '\0'; j++)
        {
            if (input[i] == diction[j])
            {
                temp1[i] = j;
                input[i] = diction[(j + key) % 26];
                break;
            }
            if (input[i] == dictionCap[j])
```

```
            {
                temp1[i] = j;
                input[i] = dictionCap[(j + key) % 26];
                break;
            }
        }
    }
    cout << "After encryption using key " << key << " :\n\n";
    for (int i = 0; input[i] != '\0'; i++)
    {
        cout << input[i];
    }
    cout << "\n\n\n**********MENU**********\n\n";
    cout << "Press 1 to decrypt the text\n2. Press 2 to
exit\n\n";
    cin >> choice;
    cin >> key;
    switch (choice)
    {
    case 1:
    {
        for (int i = 0; input[i] != '\0'; i++)
        {
            for (int j = 0; diction[j] != '\0'; j++)
            {
                if (input[i] == diction[j])
                {
                    temp2[i] = j;
                    input[i] = diction[(j - key + 26) % 26];
                    break;
                }
                if (input[i] == dictionCap[j])
                {
                    temp2[i] = j;
                    input[i] = dictionCap[(j - key + 26) % 26];
                    break;
                }
            }
        }
```

```
        cout << "After decryption using key " << key <<
" :\n\n";
        for (int i = 0; input[i] != '\0'; i++)
        {
            cout << input[i];
        }
        break;
    }
    case 2:
    {
        cout << "Thank you for using the program";
        exit(1);
    }
    default:
        cout << "Invalid Choice";
    }
}
```

```
20BCS004_Abdul Basit
Enter the Input text: GamingLaptop

Enter the key: 3
After encryption using key 3 :

JdplqjOdswrs


**********MENU**********

Press 1 to decrypt the text
2. Press 2 to exit

1
After decryption using key 3 :

GamingLaptop
PS C:\Users\Abdul Basit\Desktop\CN lab\All_In_ONE\
```

```
20BCS004_Abdul Basit
Enter the Input text: A good Ball

Enter the key: 4
After encryption using key 4 :

E kssh Fepp


**********MENU**********

Press 1 to decrypt the text
2. Press 2 to exit

1
After decryption using key 4 :

A good Ball
PS C:\Users\Abdul Basit\Desktop\CN lab\All_In_ONE\
```

Program 2 :

Transposition cipher

```
// transposition cipher
#include <bits/stdc++.h>
using namespace std;

void encrypt()
{
    cout << "Enter Key : ";
    int key;
    string s;
    cin >> key;
    cin.get();
xx:
    cout << "Enter string : ";
    getline(cin, s);
    for (char c : s)
    {
        if (!isalpha(c))
        {
            if (c != 32)
            {
                cout << "wrong input.Please try again.\n";
                goto xx;
            }
        }
    }
    int col = s.size() / key + 1;
    vector<vector<char>> encrypt(col, vector<char>(key, 'X'));
    int z = 0;
    for (int i = 0; i < col; i++)
    {
        for (int j = 0; j < key; j++)
        {
            while (s[z] == 32)
            {
                z++;
            }
```

```
            if ((s[z] >= 'a' and s[z] <= 'z') or (s[z] >= 'A'
and s[z] <= 'Z'))
            {
                encrypt[i][j] = s[z];
            }

            if (s[z] != 0)
            {
                z++;
            }
        }
    }

    string ans = "";
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < col; j++)
        {
            ans += encrypt[j][i];
        }
    }

    cout << ans << endl;
}

void decrypt()
{
    cout << "Enter Key : ";
    int key;
    string s;
    cin >> key;
    cin.get();
xx:
    cout << "Enter string : ";
    getline(cin, s);
    for (char c : s)
    {
        if (!isalpha(c))
        {
            if (c != 32)
```

```
            {
                cout << "wrong input.Please try again.\n";
                goto xx;
            }
        }
    }
    int col = s.size() / key;
    vector<vector<char>> encrypt(col, vector<char>(key));
    int z = 0;
    for (int i = 0; i < key; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (s[z] == 32)
            {
                z++;
            }

            encrypt[j][i] = s[z];
            if (s[z] != 0)
            {
                z++;
            }
        }
    }

    string ans = "";
    for (int i = 0; i < col; i++)
    {
        for (int j = 0; j < key; j++)
        {
            if (encrypt[i][j] != 'X')
            {
                ans += encrypt[i][j];
            }
        }
    }

    cout << ans << endl;
}
```

```
int main()
{
    int ch;
    while (1)
    {
    xy:
        cout << "1. Encryption.\n";
        cout << "2. Decryption.\n";
        cout << "3. Exit.\n";
        cout << "Enter your choice : ";
        cin >> ch;
        switch (ch)
        {
        case 1:
            encrypt();
            break;

        case 2:
            decrypt();
            break;

        case 3:
            return 0;

        default:
            cout << "Wrong input . Please try again \n";
            goto xy;
        }
    }
    return 0;
}
```

```
PS C:\Users\Abdul Basit> cd "c:\Users\Abdul Basit\Desktop\CN lab\All_In_ONE\All_In_ONE
tion_cipher.cpp -o transposition_cipher } ; if ($?) { .\transposition_cipher }
1. Encryption.
2. Decryption.
3. Exit.
Enter your choice : 1
Enter Key : 3
Enter string : gaming laptop
giltXanaoXmgppX
1. Encryption.
2. Decryption.
3. Exit.
Enter your choice : 2
Enter Key : 3
Enter string : giltXanaoXmgppX
gaminglaptop
```

```
1. Encryption.
2. Decryption.
3. Exit.
Enter your choice : 1
Enter Key : 2
Enter string : 123hasd
wrong input.Please try again.
Enter string : 3
wrong input.Please try again.
Enter string : hello world
hloolXelwrdX
1. Encryption.
2. Decryption.
3. Exit.
Enter your choice : 3
PS C:\Users\Abdul Basit\Desktop\CN lab\All_In_ONE\All_In_ONE\T
```

Program 3 :

Baconian Cipher

```
#include <bits/stdc++.h>
using namespace std;

// Simple Baconian cipher for letters
int main()
{
    string message;
    int option;

    map<char, string> ciphertext{{'a', "00000"}, {'b', "00001"},
{'c', "00010"}, {'d', "00011"}, {'e', "00100"}, {'f', "00101"},
{'g', "00110"}, {'h', "00111"}, {'i', "01000"}, {'j', "01001"},
{'k', "01010"}, {'l', "01011"}, {'m', "01100"}, {'n', "01101"},
{'o', "01110"}, {'p', "01111"}, {'q', "10000"}, {'r', "10001"},
{'s', "10010"}, {'t', "10011"}, {'u', "10100"}, {'v', "10101"},
{'w', "10110"}, {'x', "10111"}, {'y', "11000"}, {'z', "11001"}};

    cout << "Choose an option for the Baconian cipher:\n 1) to
encrypt \n 2) to decrypt\n";

    cin >> option;
    cin.ignore();

    char letter;

    if (option == 1)
    {

        cout << "\nEnter the message in plaintext\n\n";
        getline(cin, message);

        for (int i = 0; i < message.length(); i++) // iterates
through input, if alphabet, convert to ciphertext, if space, ig-
nores, if anything else, prints through
        {
```

```
            if (isalpha(message[i]))
            {
                letter = tolower(message[i]);
                cout << ciphertext.at(letter);
            }

            else if (isspace(message[i]))
            {
                cout << "";
            }

            else
            {
                cout << message[i];
            }
        }

        cout << "\n";
    }

    if (option == 2)
    {
        string parsed_char;

        cout << "\nEnter the message in ciphertext\n\n";
        getline(cin, message);

        for (int i = 0; i < message.length(); i = i + 5) // it-
erates through message in chunks of 5 char
        {
            for (int j = 0; j < 5; j++) // creates string
parsed_char equal to chunk of 5 char
            {
                parsed_char = parsed_char + message[i + j];
            }

            for (int k = 0; k < 26; k++) // creates alphabet,
checks if parsed_char is value of any letter key in map and
prints letter if value is in ciphertext map
            {
```

```
                int iter = 97 + k;
                letter = char(iter);
                if (ciphertext.at(letter) == parsed_char)
                {
                    cout << letter;
                }
            }
            parsed_char = "";
        }
    }
    cout << "\n";
}
```

```
Choose an option for the Baconian cipher:
 1) to encrypt
 2) to decrypt
1

Enter the message in plaintext

laptop
01011000000111110011011100111
```

```
Choose an option for the Baconian cipher:
 1) to encrypt
 2) to decrypt
2

Enter the message in ciphertext

01011000000111110011011100111
laptop
```

Program 4 :

Server-Client Socket Program

//Server.py

```
import socket
host = socket.gethostname()
port = 9999

server_socket = socket.socket()

server_socket.bind((host, port))

server_socket.listen(2)
print('waiting for connection')
conn, address = server_socket.accept()
print("Connection from: " + str(address))
while True:

    data = conn.recv(1024).decode()
    if not data:
        print('disconnected!!!')
        break
    print("from connected user: " + str(data))
    data = input(' -> ')
    conn.send(data.encode())

conn.close()
```

//Client.py

```
import socket

host = socket.gethostname()
port = 9999
client_socket = socket.socket()
client_socket.connect((host, port))

message = input(" -> ")

while message.lower().strip() != 'end':
    client_socket.send(message.encode())
    data = client_socket.recv(1024).decode()
    print('Received from server: ' + data)
    message = input(" -> ")

client_socket.close()
```

```
Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN lab/All_In_ONE/
All_In_ONE/Lab1_client-server
$ python srvr.py
waiting for connection
Connection from: ('192.168.1.4', 63063)
from connected user: hello
 -> hey
from connected user: nice
 -> 2nd message
disconnected!!!
```

```
Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN lab/All_In_O
E/All_In_ONE/Lab1_client-server
$ python client.py
 -> hello
Received from server: hey
 -> nice
Received from server: 2nd message
 -> end

Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN lab/All_In_O
```

Program 5 :

Server-Client with Substitution cipher

//SERVER.py

```
import socket as so

s = so.socket()
print("Connection made")

port = 9999
s.bind(('localhost',port))

def decrypt(msg, key):
    key = int(key)
    res = ""
    for i in msg:
        res += chr((ord(i)-key+128)%128)

    return res


s.listen(3)

print('Waiting for connections')

while True:
    c, addr = s.accept()
    while True:


        msg = c.recv(1024).decode()

        print("Msg recieved : ", msg)
        key = input("Enter key for decryption : ")
        decrypted_msg = decrypt(msg,int(key))
```

```
        if decrypted_msg == "bye":
            break


        c.send(bytes(decrypted_msg,'utf-8'))
```

//Client.py

```
import socket

c = socket.socket()
port = 9999
def encrypt(msg, key):
    key = int(key)
    res = ""
    for i in msg:
        res += chr((ord(i)+key)%128)

    return res


c.connect(('localhost',port))
while 1:
    msg = input('Enter your message : ')
    key = input('Enter your key : ')
    encrypted_msg = encrypt(msg,int(key))
    print("Cipher Text send to Server : ",encrypted_msg)
    if msg=="bye" :
        c.close()
        break

    c.send(bytes(encrypted_msg,'utf-8'))

    decrypted_msg = c.recv(1024).decode()
    print(f"Decrypted msg {decrypted_msg}")
```

```
Abdul Basit@Abdul-Basit-PC MINGW64 ~/AppData/Local/Temp/Temp1_CNL
ABALLtemp.zip/CNLABALL/p5
$ python server.py
Connection made
Waiting for connections
Msg recieved :  khoorzruog
Enter key for decryption : 3
Msg recieved :
Enter key for decryption : _
```

```
Abdul Basit@Abdul-Basit-PC MINGW64 ~/AppData/Local/Te
NLABALLtemp.zip/CNLABALL/p5
$ python client.py
Enter your message : helloworld
Enter your key : 3
Cipher Text send to Server :  khoorzruog
Decrypted msg helloworld
Enter your message : bye
Enter your key : 1
```

Program 6 :

Client-Server-Multiple Client Broadcasting

//SERVER.py

```
import socket
import threading
# Connection Data
host = '127.0.0.1'
port = 9999

# Starting Server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()

# Lists For Clients and Their Nicknames
clients = []
nicknames = []
# Sending Messages To All Connected Clients


def broadcast(message):
    for client in clients:
        client.send(message)

    print(message)

# Handling Messages From Clients


def handle(client):
    while True:
        try:
            # Broadcasting Messages
            message = client.recv(1024)
            broadcast(message)
            print(message)
        except:
            # Removing And Closing Clients
```

```
            index = clients.index(client)
            clients.remove(client)
            client.close()
            nickname = nicknames[index]
            broadcast('{} left!'.format(nickname).en-
code('ascii'))
            nicknames.remove(nickname)
            break


# Receiving / Listening Function
def receive():
    while True:
        # Accept Connection
        client, address = server.accept()
        print("Connected with {}".format(str(address)))

        # Request And Store Nickname
        client.send('NICK'.encode('ascii'))
        nickname = client.recv(1024).decode('ascii')
        nicknames.append(nickname)
        clients.append(client)

        # Print And Broadcast Nickname
        print("Nickname is {}".format(nickname))
        broadcast("{} joined!".format(nickname).encode('ascii'))
        client.send('Connected to server!'.encode('ascii'))

        # Start Handling Thread For Client
        thread = threading.Thread(target=handle, args=(client,))
        thread.start()


print("Server is listening...")
receive()
```

//CLIENT.py

```
import socket
import threading
# Choosing Nickname
nickname = input("Choose your nickname: ")

# Connecting To Server
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('127.0.0.1', 9999))

# Listening to Server and Sending Nickname

def receive():
    while True:
        try:
            # Receive Message From Server
            # If 'NICK' Send Nickname
            message = client.recv(1024).decode('ascii')
            if message == 'NICK':
                client.send(nickname.encode('ascii'))
            else:
                print(message)
        except:
            # Close Connection When Error
            print("An error occured!")
            client.close()
            break

# Sending Messages To Server
def write():
    while True:
        message = '{}: {}'.format(nickname, input(''))
        client.send(message.encode('ascii'))

# Starting Threads For Listening And Writing
receive_thread = threading.Thread(target=receive)
receive_thread.start()
write_thread = threading.Thread(target=write)
write_thread.start()
```

```
Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN lab/All_In_ONE/All_In_ONE/broadcasting
$ python server.py
Server is listening...
Connected with ('127.0.0.1', 63268)
Nickname is client1
b'client1 joined!'
Connected with ('127.0.0.1', 63269)
Nickname is client2
b'client2 joined!'
b'client1: hello1'
b'client1: hello1'
b'client2: hey2'
b'client2: hey2'
b'client2: echo'
b'client2: echo'
b'client1: broadcasting'
b'client1: broadcasting'
_
```

```
Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN lab/All_In_ONE/All_In_ONE/broadcasting
$ python client.py
Choose your nickname: client1
client1 joined!
Connected to server!
client2 joined!
hello1
client1: hello1
client2: hey2
client2: echo
broadcasting
client1: broadcasting
_
```

```
Abdul Basit@Abdul-Basit-PC MINGW
ktop/CN lab/All_In_ONE/All_In_ON
asting
$ python client.py
Choose your nickname: client2
client2 joined!
Connected to server!
client1: hello1
hey2
client2: hey2
echo
client2: echo
client1: broadcasting
_
```

Program 7 :

Check the Datatype of Input from the Client on Server

//SERVER.py

```
import socket as so
s=so.socket()

print("socket made")

port=1234

s.bind(('localhost', port))

s.listen(3)

print('waiting for connection')

while True:
    c,addr= s.accept()
    datarecv=c.recv(1024).decode()
    typeofdata='INT'
    for i in datarecv:
        if(i=='~' or i=='!' or i=='#' or i=='$' or i=='%'
or i=='^' or i=='&' or i=='*'):
            typeofdata='Special Character'
            break
    for i in datarecv:
        if(i=='.'):
            typeofdata='Float'
            break
    for i in datarecv:
        if(i>='a' and i<='z'):
            typeofdata='String'
            break
        if(i>='A' and i<='Z'):
```

```
            typeofdata='String'
            break
    if(len(datarecv)==1 and typeofdata=='String'):
        typeofdata='Character'
    print(typeofdata)
    c.send(bytes(f"{datarecv} is the data you sent whose
data type is {typeofdata} ",'utf-8'))
```

//CLIENT.py

```
import socket as so
client= so.socket()
port= 1234
id= input('Enter the data you want to check whose data
type: ')

client.connect(('localhost',port))

client.send(bytes(id,'utf-8'))

print(client.recv(1024).decode())
```

```
Install the latest PowerShell for new features and improvements! h
ttps://aka.ms/PSWindows

PS C:\Users\Abdul Basit\Desktop\CN lab\All_In_ONE\All_In_ONE\lab3s
imple> python -u "c:\Users\Abdul Basit\Desktop\CN lab\All_In_ONE\A
ll_In_ONE\lab3simple\server.py"
socket made
waiting for connection
INT
String
Float
```

```
l_In_ONE/lab3simple
$ python client.py
Enter the data you want to check whose data type: 40
40 is the data you sent whose data type is INT

Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN lab/All_In_ONE/A
l_In_ONE/lab3simple
$ python client.py
Enter the data you want to check whose data type: asd
asd is the data you sent whose data type is String

Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN lab/All_In_ONE/A
l_In_ONE/lab3simple
$ python client.py
Enter the data you want to check whose data type: 40.2
40.2 is the data you sent whose data type is Float
```

Program 8 :

Server-Client with rail-fencing cipher

//SERVER.py

```
import socket as so
s = so.socket()

print("socket made")

port = 1234

s.bind(('localhost', port))

s.listen(3)

print('waiting for connection')

while True:
    c, addr = s.accept()
    datarecv = c.recv(1024).decode()

    # print(datarecv)
    # x=datarecv.split("~")
    # cipher=x[0]
    # keyNew=int(x[1])
    print(datarecv)
    keyNew = input('Enter key to decrypt : ')
    keyNew = int(keyNew)
    # This function receives cipher-text
    # and key and returns the original
    # text after decryption

    def decryptRailFence(cipher, keyNew):

        # create the matrix to cipher
        # plain text key = rows ,
        # length(text) = columns
        # filling the rail matrix to
        # distinguish filled spaces
        # from blank ones
        rail = [['\n' for i in range(len(cipher))]
                for j in range(keyNew)]
```

```
# to find the direction
dir_down = None
row, col = 0, 0

# mark the places with '*'
for i in range(len(cipher)):
    if row == 0:
        dir_down = True
    if row == keyNew - 1:
        dir_down = False

    # place the marker
    rail[row][col] = '*'
    col += 1

    # find the next row
    # using direction flag
    if dir_down:
        row += 1
    else:
        row -= 1

# now we can construct the
# fill the rail matrix
index = 0
for i in range(keyNew):
    for j in range(len(cipher)):
        if ((rail[i][j] == '*') and
                (index < len(cipher))):
            rail[i][j] = cipher[index]
            index += 1

# now read the matrix in
# zig-zag manner to construct
# the resultant text
result = []
row, col = 0, 0
for i in range(len(cipher)):

    # check the direction of flow
    if row == 0:
        dir_down = True
    if row == keyNew-1:
        dir_down = False
```

```
            # place the marker
            if (rail[row][col] != '*'):
                result.append(rail[row][col])
                col += 1

            # find the next row using
            # direction flag
            if dir_down:
                row += 1
            else:
                row -= 1
        return ("".join(result))

    print("\nDecrypted Text: \n")
    print(decryptRailFence(datarecv, keyNew))
```

//CLIENT.py

```
import socket as so
client= so.socket()
port= 1234
key= input('Enter the key for encryption : ')
key = int(key)
client.connect(('localhost',port))


# function to encrypt a message
def encryptRailFence(text, key):

    # create the matrix to cipher
    # plain text key = rows ,
    # length(text) = columns
    # filling the rail matrix
    # to distinguish filled
    # spaces from blank ones
    rail = [['\n' for i in range(len(text))]
                for j in range(key)]

    # to find the direction
    dir_down = False
    row, col = 0, 0

    for i in range(len(text)):
```

```
        # check the direction of flow
        # reverse the direction if we've just
        # filled the top or bottom rail
        if (row == 0) or (row == key - 1):
            dir_down = not dir_down

        # fill the corresponding alphabet
        rail[row][col] = text[i]
        col += 1

        # find the next row using
        # direction flag
        if dir_down:
            row += 1
        else:
            row -= 1
    # now we can construct the cipher
    # using the rail matrix
    result = []
    for i in range(key):
        for j in range(len(text)):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    return("" . join(result))



# Driver code
text = input('Message to encrypt : ')

encrypted = encryptRailFence(text, key)

# keydec = input('Enter key to decrypt : ')
# totalSend = encrypted + '~' + keydec
client.send(bytes(encrypted,'utf-8'))
```

```
socket made
waiting for connection
gnapaiglpom t
Enter key to decrypt : 3

Decrypted Text:

gaming laptop
_
```

```
$ python client.py
Enter the key for encryption : 3
Message to encrypt : gaming laptop

Abdul Basit@Abdul-Basit-PC MINGW64 ~
ab/All_In_ONE/All_In_ONE/lab4_encryp
nt
$ _
```

Program 9 :

Server-Client with vignere cipher

//SERVER.py

```
import socket as so
s=so.socket()

print("socket made")

port=1234

s.bind(('localhost', port))

s.listen(3)

print('waiting for connection')

def generateKey(string, key):
    key = list(key)
    if len(string) == len(key):
        return(key)
    else:
        for i in range(len(string) -
                       len(key)):
            key.append(key[i % len(key)])
    return("" . join(key))

# while True:
c,addr= s.accept()
datarecv=c.recv(1024).decode()

print("Ciphertext :", datarecv)
keyNew = input('Enter key to decrypt : ')
key = generateKey(datarecv, keyNew)
def originalText(cipher_text, key):
    orig_text = []
        # print("Test 1")
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return("" . join(orig_text))
    # cipher_text =str(datarecv)
```

```
print("Original/Decrypted Text : \n")
print(originalText(datarecv, key))
c.close()
```

//CLIENT.py

```
import socket as so
client= so.socket()
port= 1234
keyword= input('Enter the key for encryption : ')
# key = int(key)
client.connect(('localhost',port))


def generateKey(string, key):
    key = list(key)
    if len(string) == len(key):
        return(key)
    else:
        for i in range(len(string) -
                       len(key)):
            key.append(key[i % len(key)])
    return("" . join(key))



def cipherText(string, key):
    cipher_text = []
    for i in range(len(string)):
        x = (ord(string[i]) +
            ord(key[i])) % 26
        x += ord('A')
        cipher_text.append(chr(x))
    return("" . join(cipher_text))







# Driver code
string = input('Message to encrypt : ')
```

```
key = generateKey(string, keyword)
cipher_text = cipherText(string,key)


# keydec = input('Enter key to decrypt : ')
# totalSend = encrypted + '~' + keydec
client.send(bytes(cipher_text,'utf-8'))
```

```
$ python server.py
socket made
waiting for connection
Ciphertext : DOEXNUEJVXUWGZDT
Enter key to decrypt : BASIT
Original/Decrypted Text :

COMPUTERNETWORKS
```

```
$ python client.py
Enter the key for encryption : BASIT
Message to encrypt : COMPUTERNETWORKS

Abdul Basit@Abdul-Basit-PC MINGW64 ~/Desktop/CN
ab/All_In_ONE/All_In_ONE/lab5_vignereCipher_serv
rClient
$ _
```

Program 10 :

Play Fair cipher

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
#define MATRIX_SIZE 5

vector<vector<char>> cipher_matrix(string key)
{
    const int LIST_SIZE = 25;
    vector<char> letter_list;
    vector<char> result_list;
    for (char letter = 'a'; letter <= 'z'; letter++)
    {
        if (letter != 'j')
            letter_list.push_back(letter);
    }
    for (int i = 0; i < key.length(); i++)
    {
        for (int j = 0; j < LIST_SIZE; j++)
        {
            if (key[i] == letter_list[j])
            {
                result_list.push_back(key[i]);
                letter_list[j] = '-';
                break;
            }
        }
    }
    for (int i = 0; i < LIST_SIZE; i++)
    {
        if (letter_list[i] != '-')
            result_list.push_back(letter_list[i]);
```

```
    }

    vector<vector<char>> matrix;
    vector<char> temp_vector;
    for (int i = 0; i < result_list.size(); i++)
    {
        temp_vector.push_back(result_list[i]);
        if ((i + 1) % MATRIX_SIZE == 0)
        {
            matrix.push_back(temp_vector);
            temp_vector.clear();
            //          cout << temp_vector.size();
        }
    }
    return matrix;
}
void print_matrix(vector<vector<char>> matrix)
{
    for (int i = 0; i < matrix.size(); i++)
    {
        for (int j = 0; j < matrix[i].size(); j++)
        {
            cout << matrix[i][j] << '\t';
        }
        cout << endl;
    }
}
int shift_left_up(int position)
{
    return (position + 4) % MATRIX_SIZE;
}

int shift_right_down(int position)
{
    return (position + 1) % MATRIX_SIZE;
```

```
}
string cipher_decipher(string input, bool mode, vec-
tor<vector<char>> matrix)
{
    string output = "";
    string init = "";
    char first = input[0];
    char second = input[1];
    int first_row = 0, first_col = 0;
    int second_row = 0, second_col = 0;
    for (int i = 0; i < MATRIX_SIZE; i++)
    {
        for (int j = 0; j < MATRIX_SIZE; j++)
        {
            if (first == matrix[i][j])
            {
                first_row = i;
                first_col = j;
            }
            if (second == matrix[i][j])
            {
                second_row = i;
                second_col = j;
            }
        }
    }
    if (first_row == second_row)
    {
        //      shift_left_upshift_right_down
        if (mode == true)
        {
            output = init + ma-
trix[first_row][shift_right_down(first_col)] + matrix[sec-
ond_row][shift_right_down(second_col)];
        }
```

```
        else if (mode == false)
        {
            output = init + ma-
trix[first_row][shift_left_up(first_col)] + matrix[sec-
ond_row][shift_left_up(second_col)];
        }
    }
    else if (first_col == second_col)
    {
        if (mode == true)
        {
            output = init + ma-
trix[shift_right_down(first_row)][first_col] + ma-
trix[shift_right_down(second_row)][second_col];
        }
        else if (mode == false)
        {
            output = init + ma-
trix[shift_left_up(first_row)][first_col] + ma-
trix[shift_left_up(second_row)][second_col];
        }
    }
    else
    {
        output = init + matrix[first_row][second_col] +
matrix[second_row][first_col];
    }

    return output;
}

int main()
{

    string input = ""; // tessinx sample
```

```
    string output = "";
    string key = ""; // committed sample
    cout << "Input: ";
    // cin >> input;
    getline(cin, input);
    cout << "Key: ";
    // cin >> key;
    getline(cin, key);
    string mode_str = "";
    bool mode = true; // encryption by default
    cout << "Choose 'en' for encrypt and 'de' for decryp-
tion (encryption by default): ";
    cin >> mode_str;
    if (mode_str == "de")
    {
        mode = false;
    }
    else
    {
        mode = true;
    }
    // input.erase(remove_if(input.begin(), input.end(), '
'), input.end());

    // trim whitespace of input
    string temp_str = "";
    for (int i = 0; i < input.length(); i++)
    {
        if (input[i] != ' ')
            temp_str += input[i];
    }
    input = temp_str;

    // trim whitespace of key
    temp_str = "";
```

```
    for (int i = 0; i < key.length(); i++)
    {
        if (key[i] != ' ')
            temp_str += key[i];
    }
    key = temp_str;
    if ((input.length() % 2) == 1)
        input += "x";
    vector<vector<char>> matrix = cipher_matrix(key);
    string process_pair = "";
    //   print_matrix(matrix);
    while (true)
    {
        if (input.length() == 1)
        {
            process_pair = input + "x";
        }
        else
        {
            process_pair = input.substr(0, 2);
            if (process_pair[0] == process_pair[1])
            {
                input = process_pair.substr(1, 1) + input;
                process_pair = process_pair.substr(0, 1) +
"x";
                //            cout << process_pair;
            }
        }
        process_pair = cipher_decipher(process_pair, mode,
matrix);
        output += process_pair;
        //      cout << process_pair << '\t' << output ;
        if (input.length() > 2)
        {
            input = input.substr(2);
```

```
        }
        else
            break;
    }
    cout << output;
    return 0;
}
```

```
PS C:\Users\Abdul Basit\Desktop\CN lab\All_In_ONE\All_In_ONE\playfairCipher>
CN lab\All_In_ONE\All_In_ONE\playfairCipher\" ; if ($?) { g++ playFairNew.cpp
layFairNew }
Input: instruments
Key: monarchy
Choose 'en' for encrypt and 'de' for decryption (encryption by default): en
gatlmzclrqxa
```

```
Input: gatlmzclrqxa
Key: monarchy
Choose 'en' for encrypt and 'de' for decryption (encryption by default): de
instrumentsx
```