CEN 593 – Computer Network Laboratory File

BTech Computer Engineering
Vth Semester

Submitted by:
Aiman Fatima (20BCS008)

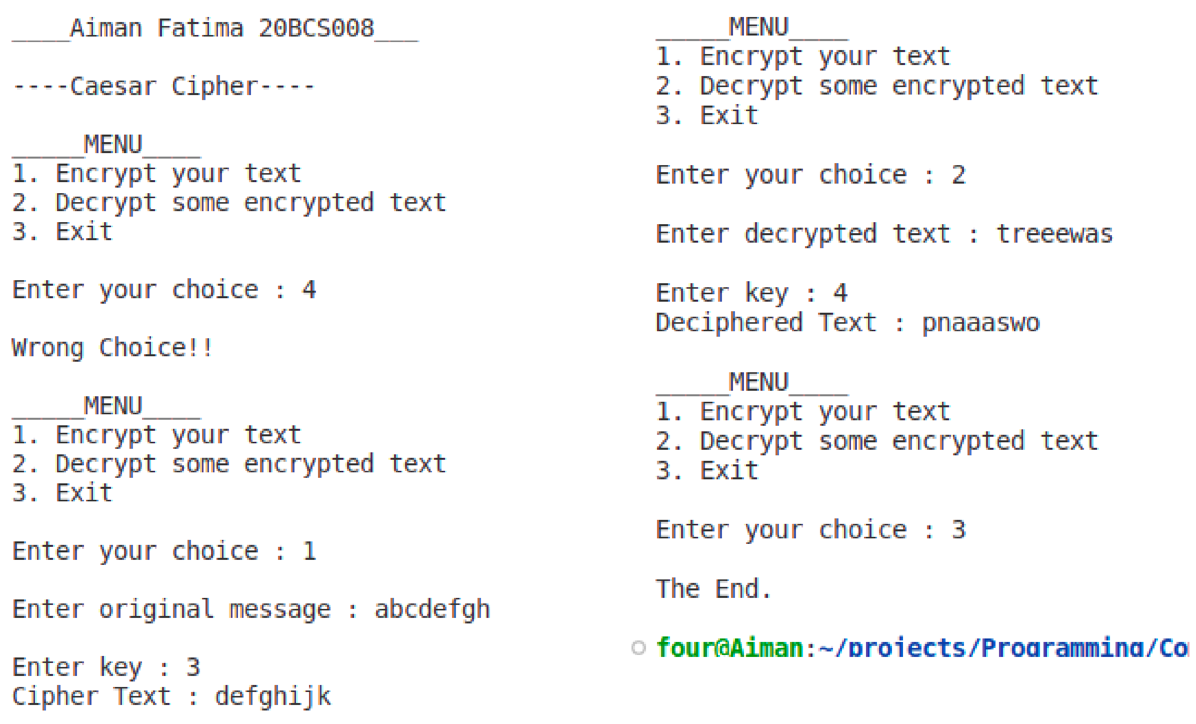
Submitted to:
Professor M Amjad
Mr. Hannan Mansoor

Department of Computer Engineering,
Faculty of Engineering & Technology,
Jamia Millia Islamia, New Delhi
2022

Index

```
------------------------------ Program 1 ---------------------------------
----------------------------- Caesar Cipher ------------------------------
#include <bits/stdc++.h>
using namespace std;

string encrypt(string msg, int key)
{
    string res = "";

    for (auto &i : msg)
    {
        res += (((i - 'a' + key) % 26) + 'a');
    }
    return res;
}

string decrypt(string msg, int key)
{
    string res = "";

    for (auto &i : msg)
    {
        res += ('a' + ((i - 'a' - key + 26) % 26));
    }
    return res;
}

int main()
{
    cout << "\n____Aiman Fatima 20BCS008___\n";
    cout << "\n----Caesar Cipher----\n";
    string msg;
    string hidden_msg, clear_msg;
    int key;
    while (1)
    {
        cout << "\n_____MENU____\n";
        cout << "1. Encrypt your text\n";
        cout << "2. Decrypt some encrypted text\n";
        cout << "3. Exit\n";
        cout << "\nEnter your choice : ";
        int choice;
        cin >> choice;
        msg.clear();
        switch (choice)
        {
        case 1:
            cout << "\nEnter original message : ";
            getchar();
            getline(cin, msg);
            cout << "\nEnter key : ";
            cin >> key;
            hidden_msg = encrypt(msg, key);
            cout << "Cipher Text : " << hidden_msg << "\n";
            break;
        case 2:
            cout << "\nEnter decrypted text : ";
            getchar();
            getline(cin, msg);
            cout << "\nEnter key : ";
            cin >> key;
            clear_msg = decrypt(msg, key);
            cout << "Deciphered Text : " << clear_msg << "\n";
            break;
        case 3:
            cout << "\nThe End.\n\n";
            return 0;
        default:
            cout << "\nWrong Choice!!\n";
        }
    }
}
```

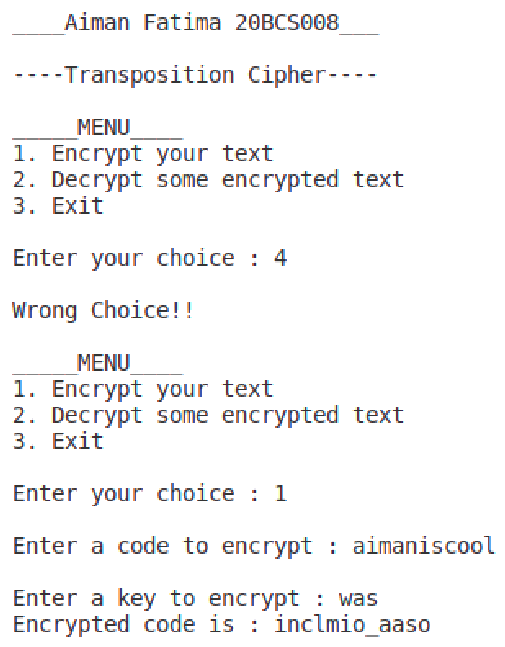
___________Output___________



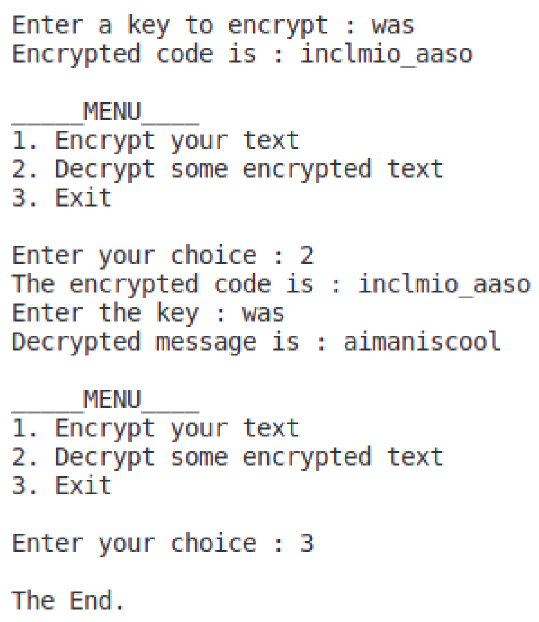
```
____Aiman Fatima 20BCS008___

----Caesar Cipher----

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 4

Wrong Choice!!

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 1

Enter original message : abcdefgh

Enter key : 3
Cipher Text : defghijk

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 2

Enter decrypted text : treeewas

Enter key : 4
Deciphered Text : pnaaaswo

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 3

The End.

○ four@Aiman:~/projects/Programming/Co
```

```
------------------------------- Program 2 ----------------------------------
------------------------------ Transposition Cipher ---------------------------------
#include <iostream>
#include <vector>
#include <string>
#include <cctype>
#include <map>

using namespace std;

map<int, int> keyMap;

void setPermutationOrder(string key)
{
    for (int i = 0; i < key.length(); i++)
    {
        keyMap[key[i]] = i;
    }
}

// Encryption
string encrypt(string msg, string key, vector<char> &encrypted)
{
    int row, col, j;
    col = key.length();
    row = msg.length() / col;

    if (msg.length() % col)
        row += 1;
    vector<vector<char>> matrix(row, vector<char>(col, '#'));

    for (int i = 0, k = 0; i < row; i++)
    {
        for (int j = 0; j < col;)
        {
            if (msg[k] == '\0')
            {
                matrix[i][j] = '_';
                j++;
            }

            if (isalpha(msg[k]) || msg[k] == ' ')
            {
                matrix[i][j] = msg[k];
                j++;
            }
            k++;
        }
    }

    for (map<int, int>::iterator ii = keyMap.begin(); ii != keyMap.end(); ++ii)
    {
        j = ii->second;
        for (int i = 0; i < row; i++)
        {
            if (isalpha(matrix[i][j]) || matrix[i][j] == ' ' || matrix[i][j] == '_')
            {
                encrypted.push_back(matrix[i][j]);
            }
        }
    }
    cout << "Encrypted code is : ";
    return string(encrypted.begin(), encrypted.end());
}

// Decryption
string decrypt(vector<char> &cipher, string key)
{
    int col = key.length();
    int row = cipher.size() / col;
    int n = cipher.size();
    cout << "The encrypted code is : ";
    for (int i = 0; i < n; i++)
    {
        cout << cipher[i];
    }
    string kk;
    cout << endl;
    cout << "Enter the key : ";
    cin >> kk;
    if (kk != key)
    {
        cout << "Wrong key entered. Try again.";
        return "";
    }
    vector<vector<char>> cipherMat(row, vector<char>(col, '#'));
    for (int j = 0, k = 0; j < col; j++)
        for (int i = 0; i < row; i++)
            cipherMat[i][j] = cipher[k++];
    int index = 0;
    for (map<int, int>::iterator ii = keyMap.begin(); ii != keyMap.end(); ++ii)
        ii->second = index++;
    vector<vector<char>> decCipher(row, vector<char>(col, '#'));
    map<int, int>::iterator ii = keyMap.begin();
    int k = 0;
    for (int l = 0, j; key[l] != '\0'; k++)
    {
        j = keyMap[key[l++]];
        for (int i = 0; i < row; i++)
        {
            decCipher[i][k] = cipherMat[i][j];
        }
    }
    string msg = "";
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
        {
            if (decCipher[i][j] != '_')
                msg += decCipher[i][j];
        }
    }
    cout << "Decrypted message is : ";
    return msg;
}

int main()
{
    cout << "\n____Aiman Fatima 20BCS008___\n";
    cout << "\n----Transposition Cipher----\n";
    string code, key;
    vector<char> encrypted, e;
    int flag = 0;
    int choice;
    while (1)
    {
        cout << "\n_____MENU____\n";
        cout << "1. Encrypt your text\n";
        cout << "2. Decrypt some encrypted text\n";
        cout << "3. Exit\n";
        cout << "\nEnter your choice : ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            cout << "\nEnter a code to encrypt : ";
            // fflush(stdin);
            getchar();
            getline(cin, code);
            cout << "\nEnter a key to encrypt : ";
            cin >> key;
            keyMap.clear();
            setPermutationOrder(key);
            cout << encrypt(code, key, encrypted) << endl;
            e = encrypted;
            encrypted.clear();
            flag = 1;
            break;
        case 2:
            if (flag == 0)
            {
                cout << "\nYou have nothing to decrypt\n";
            }
            else
            {
                cout << decrypt(e, key) << endl;
            }
            break;
        case 3:
            cout << "\nThe End.\n\n";
            return 0;
        default:
            cout << "\nWrong Choice!!\n";
        }
    }
    return 0;
}
```

___________Output___________



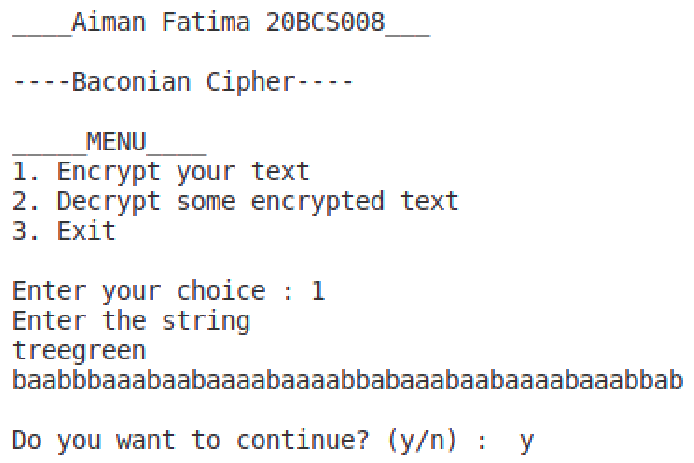
```
 ____Aiman Fatima 20BCS008___

 ----Transposition Cipher----

 _____MENU____
 1. Encrypt your text
 2. Decrypt some encrypted text
 3. Exit

 Enter your choice : 4

 Wrong Choice!!

 _____MENU____
 1. Encrypt your text
 2. Decrypt some encrypted text
 3. Exit

 Enter your choice : 1

 Enter a code to encrypt : aimaniscool

 Enter a key to encrypt : was
 Encrypted code is : inclmio_aaso
```



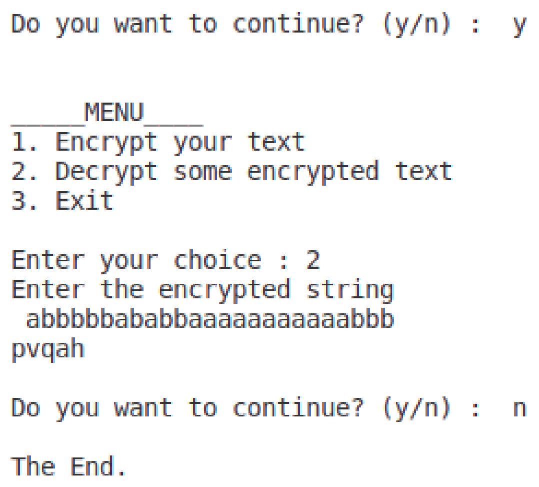
```
Enter a key to encrypt : was
Encrypted code is : inclmio_aaso

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 2
The encrypted code is : inclmio_aaso
Enter the key : was
Decrypted message is : aimaniscool

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 3

The End.
```

```
------------------------------- Program 3 ---------------------------------
------------------------------ Baconian Cipher ----------------------------------
#include <iostream>
#include <vector>
#include <map>
#include <string>

using namespace std;
map<char, string> cipherMap{{'a', "aaaaa"}, {'b', "aaaab"}, {'c', "aaaba"}, {'d', "aaabb"},
{'e', "aabaa"}, {'f', "aabab"}, {'g', "aabba"}, {'h', "aabbb"}, {'i', "abaaa"}, {'j',
"abaab"}, {'k', "ababa"}, {'l', "ababb"}, {'m', "abbaa"}, {'n', "abbab"}, {'o', "abbba"},
{'p', "abbbb"}, {'q', "baaaa"}, {'r', "baaab"}, {'s', "baaba"}, {'t', "baabb"}, {'u',
"babaa"}, {'v', "babab"}, {'w', "babba"}, {'x', "babbb"}, {'y', "bbaaa"}, {'z', "bbaab"}};

int main()
{
    cout << "\n____Aiman Fatima 20BCS008___\n";
    cout << "\n----Baconian Cipher----\n";
    string code;
    int option;
    while (1)
    {
        cout << "\n_____MENU____\n";
        cout << "1. Encrypt your text\n";
        cout << "2. Decrypt some encrypted text\n";
        cout << "3. Exit\n";
        cout << "\nEnter your choice : ";
        cin >> option;
        cin.ignore();
        char temp;
        if (option == 1)
        {
            cout << "Enter the string\n";
            getline(cin, code);
            for (int i = 0; i < code.length(); i++)
            {
                if (isalpha(code[i]))
                {
                    temp = tolower(code[i]);
                    cout << cipherMap.at(temp);
                }
                else if (isspace(code[i]))
                {
                    cout << " ";
                }
                else
                {
                    cout << code[i];
                }
            }
            cout << endl;
        }
        if (option == 2)
        {
            string s;
            cout << "Enter the encrypted string\n ";
            getline(cin, code);
            for (int i = 0; i < code.length(); i++)
            {
                code[i] = tolower(code[i]);
            }
            for (int i = 0; i < code.length(); i = i + 5)
            {
                for (int j = 0; j < 5; j++)
                {
                    s = s + code[i + j];
                }
                for (auto m : cipherMap)
                {
                    if (m.second == s)
                    {
                        cout << m.first;
                    }
                }
                s = "";
            }
            cout << endl;
        }
        if (option == 3)
        {
            cout << "\nThe End.\n\n";
            break;
        }
        cout << "\nDo you want to continue? (y/n) :  ";
        cin >> temp;
        if (temp == 'n')
        {
            cout << "\nThe End.\n\n";
            break;
        }
        cout << endl;
    }
    cout << endl;
    return 0;
}
```

___________Output___________



```
____Aiman Fatima 20BCS008___

----Baconian Cipher----

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 1
Enter the string
treegreen
baabbbaaabaabaaaabaaaabbabaaabaabaaaabaaabbab

Do you want to continue? (y/n) :  y
```



```
Do you want to continue? (y/n) :  y


_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 2
Enter the encrypted string
 abbbbbababbaaaaaaaaaaabbb
pvqah

Do you want to continue? (y/n) :  n

The End.
```

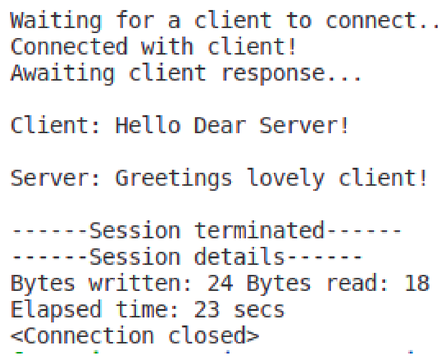
```
+++++++++++++++++++++ Socket Programs ++++++++++++++++++++

For C++
To Run ::
g++ server.cpp -o server
g++ client.cpp -o client
./server port_address
./client ip_address port_address

For Node
To Run::
npm install
node server.js
node client.js

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


------------------------------- Program 4 ----------------------------------
------------------------ Server-Client Socket Program  -------------------------

---------|| server.cpp ||-----------
// Server side
#include <iostream>
#include <string>
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;

// function to check palindrome
bool palindrome(string s)
{
    for (int i = 0; i <= s.size() / 2; i++)
    {
        if (s[i] != s[s.size() - 1 - i])
            return false;
    }
    return true;
}

// main function
int main(int argc, char *argv[])
{
    // for the server, we only need to specify a port number
    if (argc != 2)
    {
        cerr << "Usage: port" << endl;
        exit(0);
    }
    // grab the port number
    int port = atoi(argv[1]);

    // setup a socket and connection tools
    sockaddr_in servAddr;
    bzero((char *)&servAddr, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(port);

    // open stream oriented socket with internet address
    // also keep track of the socket descriptor
    int serverSd = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSd < 0)
    {
        cerr << "Error establishing the server socket" << endl;
        exit(0);
    }
    // binding the socket to its local address
    int bindStatus = bind(serverSd, (struct sockaddr *)&servAddr,
                          sizeof(servAddr));
    if (bindStatus < 0)
    {
        cerr << "Error binding socket to local address" << endl;
        exit(0);
    }
    cout << "Waiting for a client to connect..." << endl;
    // listen for up to 5 requests at a time
    listen(serverSd, 5);

    // receive a request from client using accept
    // we need a new address to connect with the client
    sockaddr_in newSockAddr;
    socklen_t newSockAddrSize = sizeof(newSockAddr);
    // accept, create a new socket descriptor to
    // handle the new connection with client
    int newSd = accept(serverSd, (sockaddr *)&newSockAddr, &newSockAddrSize);
    if (newSd < 0)
    {
        cerr << "Error accepting request from client!" << endl;
        exit(1);
    }
    cout << "Connected with client!" << endl;

    // keeping track of the session time and amount of data sent
    struct timeval start1, end1;
    gettimeofday(&start1, NULL);
    int bytesRead, bytesWritten = 0;

    // buffer to send and receive messages with
    char msg[1500];
    // variables
    string received_string;
    bool checker;
    string data;

    cout << "Awaiting client response..." << endl;
    memset(&msg, 0, sizeof(msg)); // clear the buffer
    bytesRead += recv(newSd, (char *)&msg, sizeof(msg), 0);
    received_string = msg;
    cout << "\nClient: " << received_string << endl;
    cout << "\nServer: ";
    getline(cin, data);
    // checking for palindrom
    // checker = palindrome(received_string);

    memset(&msg, 0, sizeof(msg)); // clear the buffer
    // if (checker)
    //     data = "Data received is a palindrome\n";
    // else
    //     data = "Data received is not a palindrome\n";

    // send the message to client
    strcpy(msg, data.c_str());
    bytesWritten += send(newSd, (char *)&msg, strlen(msg), 0);

    // closing the socket descriptors
    gettimeofday(&end1, NULL);
    close(newSd);
    close(serverSd);
    cout << "\n------Session terminated------" << endl;
    cout << "------Session details------" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "<Connection closed>" << endl;
    return 0;
}
```

```
---------|| client.cpp ||-----------
// Client side
#include <iostream>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;

int main(int argc, char *argv[])
{
    //  two things req: ip address and port number respectively
    if (argc != 3)
    {
        cerr << "Usage: ip_address port" << endl;
        exit(0);
    }

    // grab the IP address and port number
    char *serverIp = argv[1];
    int port = atoi(argv[2]);

    // setup a socket and connection tools
    struct hostent *host = gethostbyname(serverIp);
    sockaddr_in sendSockAddr;
    bzero((char *)&sendSockAddr, sizeof(sendSockAddr));
    sendSockAddr.sin_family = AF_INET;
    sendSockAddr.sin_addr.s_addr = inet_addr(inet_ntoa(*(struct in_addr *)*host-
>h_addr_list));
    sendSockAddr.sin_port = htons(port);
    int clientSd = socket(AF_INET, SOCK_STREAM, 0);
    // try to connect...
    int status = connect(clientSd, (sockaddr *)&sendSockAddr, sizeof(sendSockAddr));
    if (status < 0)
    {
        cout << "Error connecting to socket!" << endl;
        exit(EXIT_FAILURE);
    }
    cout << "Connected to the server!" << endl;

    // keeping track of the session time and amount of data sent
    int bytesRead, bytesWritten = 0;
    struct timeval start1, end1;
    gettimeofday(&start1, NULL);

    // creating a message buffer
    char msg[1500];
    string data;

    cout << "Client: ";
    // taking input from client
    getline(cin, data);
    memset(&msg, 0, sizeof(msg)); // clear the buffer

    // sending data to the server
    strcpy(msg, data.c_str());
    bytesWritten += send(clientSd, (char *)&msg, strlen(msg), 0);

    cout << "\nAwaiting server response..." << endl;
    memset(&msg, 0, sizeof(msg)); // clear the buffer

    // receiving server's message
    bytesRead += recv(clientSd, (char *)&msg, sizeof(msg), 0);
    cout << "\nServer: " << msg << endl;

    // closing the socket descriptors
    gettimeofday(&end1, NULL);
    close(clientSd);
    cout << "\n------Session terminated------" << endl;
    cout << "------Session details------" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "<Connection closed>" << endl;
    return 0;
}
```

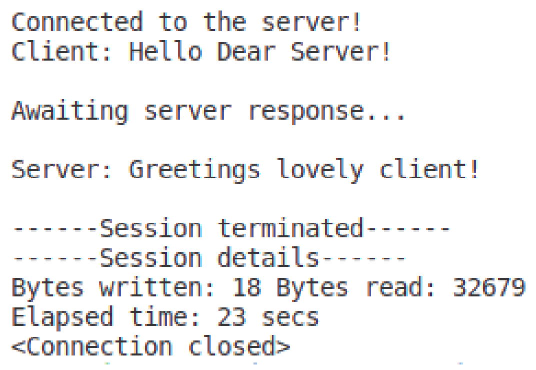
___________Output___________

Server :



```
Waiting for a client to connect...
Connected with client!
Awaiting client response...

Client: Hello Dear Server!

Server: Greetings lovely client!

------Session terminated------
------Session details------
Bytes written: 24 Bytes read: 18
Elapsed time: 23 secs
<Connection closed>
```

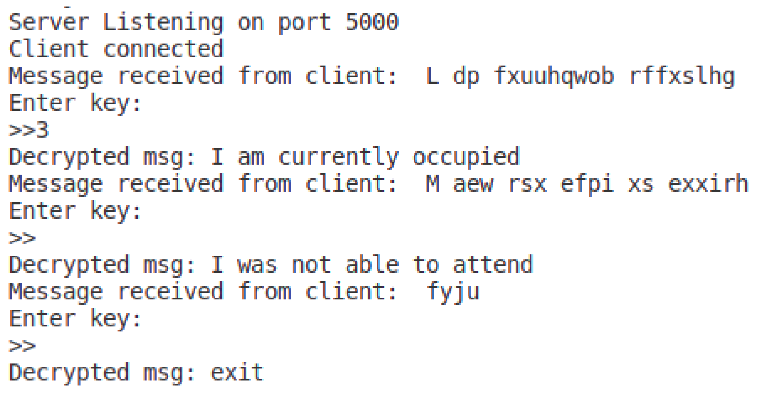
Client :



```
Connected to the server!
Client: Hello Dear Server!

Awaiting server response...

Server: Greetings lovely client!

------Session terminated------
------Session details------
Bytes written: 18 Bytes read: 32679
Elapsed time: 23 secs
<Connection closed>
```

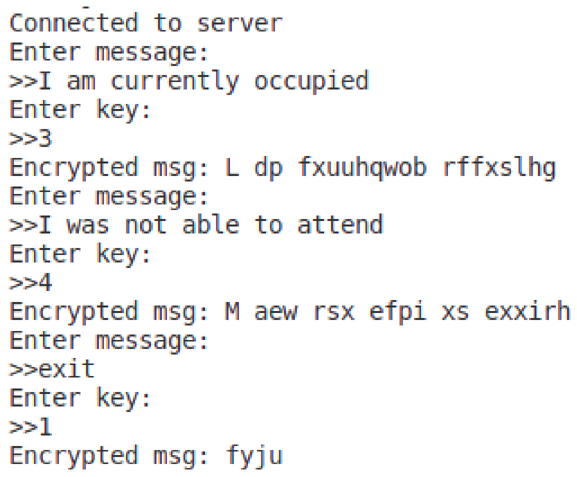
```
------------------------------- Program 5 ----------------------------------
----------------- Server-Client with Substitution Cipher ---------------------

---------|| server.js ||-----------
const { WebSocketServer } = require("ws");
const wss = new WebSocketServer({ port: 5000 });
const readline = require("readline");

let rl;

console.log("Server Listening on port 5000");

wss.on("connection", (ws) => {
  console.log("Client connected");

  ws.on("message", (message) => {
    message = message.toString();
    console.log("Message received from client: ", message);
    console.log("Enter key: ");
    getKey(message);
  });

  ws.on("close", () => {
    console.log("Client disconnected");
  });
});

//to accept key
const getKey = (message) => {
  if (rl) rl.close(); //closing previous readline

  //creating new readline
  rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });
  rl.question(">>", (key) => {
    decrypted_msg = decrypt(message, key);
  });
};

const decrypt = (message, key) => {
  let decrypted = "";
  key = Number(key);
  let code, new_char;

  for (i = 0; i < message.length; i++) {
    code = message.charCodeAt(i);
    new_char = message[i];
    if (code >= 65 && code <= 90) {
      new_char = String.fromCharCode(((code - 65 - key + 26) % 26) + 65);
      // console.log("New code", ((code - 65 - key) % 26) + 65);
    } else if (code >= 97 && code <= 122) {
      new_char = String.fromCharCode(((code - 97 - key + 26) % 26) + 97);
      // console.log("New code", ((code - 97 - key) % 26) + 97);
    }
    new_char = new_char.toString();
    decrypted += new_char;
  }
  console.log("Decrypted msg: " + decrypted);
  return decrypted;
};

---------|| client.js ||-----------
const WebSocket = require("ws");
const socket = new WebSocket("ws://localhost:5000");
const readline = require("readline");

let rl;

socket.on("open", () => {
  console.log("Connected to server");
  sendmsg1();
});

socket.on("error", () => {
  console.log("Error connecting to server");
});

//to accept message
const sendmsg1 = () => {
  console.log("Enter message: ");
  if (rl) rl.close(); //closing previous readline

  //creating new readline
  rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });
  rl.question(">>", (message) => {
    console.log("Enter key: ");
    sendmsg2(message);
  });
};

//to accept key
const sendmsg2 = (message) => {
  if (rl) rl.close(); //closing previous readline

  //creating new readline
  rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });
  rl.question(">>", (key) => {
    encrypted_msg = encrypt(message, key);
    socket.send("" + encrypted_msg);
    sendmsg1();
  });
};

const encrypt = (message, key) => {
  let encrypted = "";
  key = Number(key);
  let code, new_char;

  for (i = 0; i < message.length; i++) {
    code = message.charCodeAt(i);
    new_char = message[i];
    if (code >= 65 && code <= 90) {
      new_char = String.fromCharCode(((code - 65 + key) % 26) + 65);
    } else if (code >= 97 && code <= 122) {
      new_char = String.fromCharCode(((code - 97 + key) % 26) + 97);
    }
    encrypted += new_char;
  }
  console.log("Encrypted msg: " + encrypted);
  return encrypted;
};
```

```
___________Output___________
Server :
```



```
Server Listening on port 5000
Client connected
Message received from client:  L dp fxuuhqwob rffxslhg
Enter key:
>>3
Decrypted msg: I am currently occupied
Message received from client:  M aew rsx efpi xs exxirh
Enter key:
>>
Decrypted msg: I was not able to attend
Message received from client:  fyju
Enter key:
>>
Decrypted msg: exit
```

```
Client :
```

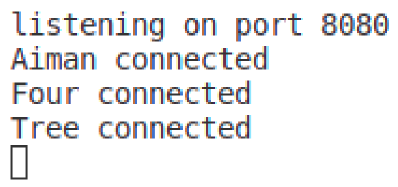


```
Connected to server
Enter message:
>>I am currently occupied
Enter key:
>>3
Encrypted msg: L dp fxuuhqwob rffxslhg
Enter message:
>>I was not able to attend
Enter key:
>>4
Encrypted msg: M aew rsx efpi xs exxirh
Enter message:
>>exit
Enter key:
>>1
Encrypted msg: fyju
```

```
------------------------------- Program 6 ----------------------------------
------- Client-Server-Multiple Client Broadcasting Socket Program -------------

---------|| server.js ||-----------
const { WebSocketServer } = require("ws");
const wss = new WebSocketServer({ port: 8080 });
const { v4: uuid } = require("uuid");

console.log("listening on port 8080");

//store all the clients
const clients = {};

/**
 * @description
 * 1. wss.on('connection') is triggered when a client connects to the server
 * 2. ws.on('message') is triggered when the server receives a message from a client
 * 3. ws.on('close') is triggered when the server receives a close event from a client
 */
wss.on("connection", (ws) => {
  //generate a unique id for the client
  const id = uuid();
  clients[id] = { name: null, ws };

  const payload = {
    method: "get id",
    id,
  };
  //send the id to the client
  ws.send(JSON.stringify(payload));

  ws.on("message", (data) => {
    //parse the data received from the client
    const payload = JSON.parse(data);
    const method = payload.method;

    //if method is new user, save the name of the client
    if (method === "new user") {
      const name = payload.name;
      const userId = payload.id;
      clients[userId].name = name;
      console.log(`${name} connected`);
      //send a message to all other clients that a new user has joined
      broadcastMsg(userId, "has joined the chat");
    }
    //if method is chat message, broadcast the message to all other clients
    else if (method === "chat message") {
      const msg = payload.msg;
      const userId = payload.id;
      broadcastMsg(userId, msg);
    }
  });

  ws.on("close", () => {
    //id remain persisted because of closure
    const name = clients[id].name;

    //when client disconnected without sending name
    if (!name) return;

    console.log(`${name} disconnected`);

    //remove the disconnected client from the clients object
    delete clients[id];

    //broadcast the disconnection to all other clients
    Object.keys(clients).forEach((clientId) => {
      const payload = {
        method: "chat message",
        msg: "has left the chat",
        name,
      };
      clients[clientId].ws.send(JSON.stringify(payload));
    });
  });
});

/**
 * @description
 * Method for broadcasting a message to all other clients
 * @param {String} userId
 * id of the client who sent the message
 * @param {String} msg
 * message to be broadcast
 */
const broadcastMsg = (userId, msg) => {
  //get the name of the sender
  const sender = clients[userId].name;

  //filter out all other clients except the sender
  const otherClientIds = Object.keys(clients).filter((id) => id !== userId);

  //send the msg to all other clients
  otherClientIds.forEach((clientId) => {
    const payload = {
      method: "chat message",
      msg,
      name: sender,
    };
    clients[clientId].ws.send(JSON.stringify(payload));
  });
};
```

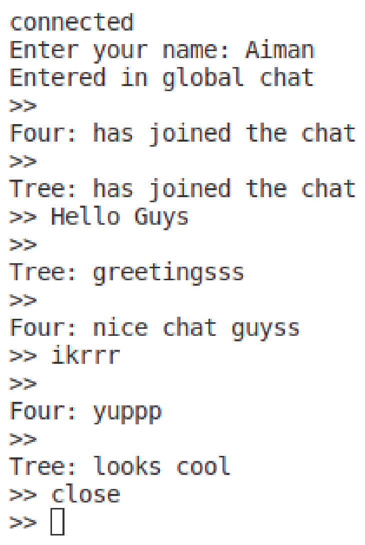
```
---------|| client.js ||-----------
const WebSocket = require("ws");
const socket = new WebSocket("ws://localhost:8080");
const readline = require("readline");

//singleton readline interface
let rl;
let isUsernameSaved = false;
let userId;

/**
 * @description
 * 1. socket.on('open') is triggered when the client is connected to the server
 * 2. socket.on('message') is triggered when the client receives a message from the server
 * 3. socket.on('error') is triggered when the client encounters an error
 * 4. socket.on('close') is triggered when the server is disconnected from the client
 */
socket.on("open", () => {
  console.log("connected");
});

socket.on("message", (data) => {
  //parse the data to get the payload
  const payload = JSON.parse(data);
  const method = payload.method;

  //if the method is new user, save the id for future requests
  if (method === "get id") {
    userId = payload.id;

    //get username after getting the id
    getName();
  }
  //if the method is chat message, print the message
  else if (method === "chat message") {
    console.log(`\n${payload.name}: ${payload.msg}`);

    /**
     * when multiple users joins at the same time,
     * then the streams of other users gets forwarded
     * and name of other users cannot be saved so,
     * it keeps asking for name on every msg received
     */
    if (!isUsernameSaved) getName();
    // otherwise, message prompt will continue to be shown for every msg received
    else sendMsg();
  }
});

socket.on("error", () => {
  console.log("Error Connecting to Server");
});

socket.on("close", () => {
  console.log("\nServer Disconnected");
  process.exit();
});

/**
 * @description
 * Method for sending messages to server
 */
```

```
const sendMsg = () => {
  //close the previous readline interface
  if (rl) rl.close();

  //create a new readline interface
  rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });
  rl.question(">> ", (msg) => {
    const payload = {
      method: "chat message",
      msg,
      id: userId,
    };
    socket.send(JSON.stringify(payload));

    //message prompt will continue to be shown for every msg sent for continuous chat
    sendMsg();
  });
};

/**
 * @description
 * Method for getting username from user
 */
const getName = () => {
  //close the previous readline interface
  if (rl) rl.close();

  //create a new readline interface
  rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout,
  });
  rl.question("Enter your name: ", (name) => {
    const payload = {
      method: "new user",
      name,
      id: userId,
    };
    socket.send(JSON.stringify(payload));

    //save the name
    isUsernameSaved = true;

    console.log("Entered in global chat");
    //start the message prompt for this user
    sendMsg();
  });
};

___________Output___________
Server :
```



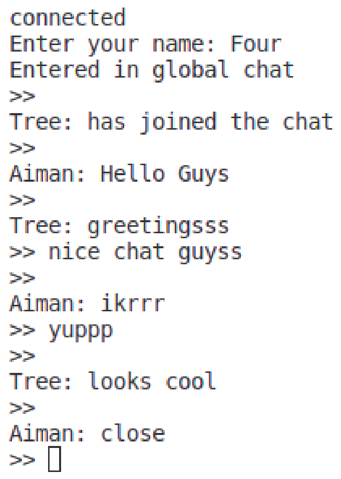
```
listening on port 8080
Aiman connected
Four connected
Tree connected
```

Client 1:



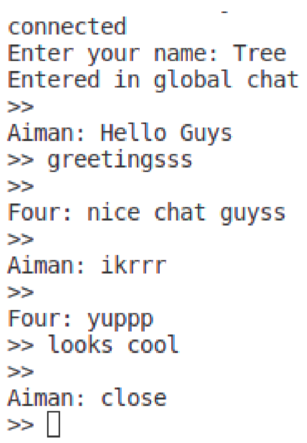
```
connected
Enter your name: Aiman
Entered in global chat
>>
Four: has joined the chat
>>
Tree: has joined the chat
>> Hello Guys
>>
Tree: greetingsss
>>
Four: nice chat guyss
>> ikrrr
>>
Four: yuppp
>>
Tree: looks cool
>> close
>>
```

Client 2:



```
connected
Enter your name: Four
Entered in global chat
>>
Tree: has joined the chat
>>
Aiman: Hello Guys
>>
Tree: greetingsss
>> nice chat guyss
>>
Aiman: ikrrr
>> yuppp
>>
Tree: looks cool
>>
Aiman: close
>>
```

Client 3:



```
connected
Enter your name: Tree
Entered in global chat
>>
Aiman: Hello Guys
>> greetingsss
>>
Four: nice chat guyss
>>
Aiman: ikrrr
>>
Four: yuppp
>> looks cool
>>
Aiman: close
>>
```

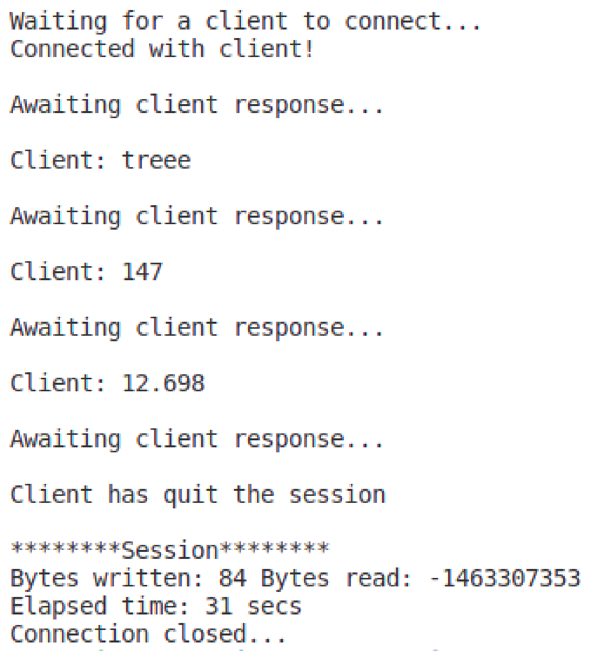
```
------------------------------- Program 7 ----------------------------------
----------- Check the Datatype of Input from the Client on the Server ----------------

---------|| server.cpp ||-----------
#include <iostream>
#include <string>
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;
// Server side

int isint(char a[])
{
    int len = strlen(a);
    int minus = 0;
    int dsum = 0;
    for (int i = 0; i < len; i++)
    {
        if (isdigit(a[i]) != 0)
            dsum++;
        else if (a[i] == '-')
            minus++;
    }
    if (dsum + minus == len)
        return 1;
    else
        return 0;
}
int isfloat(char a[])
{
    int len = strlen(a);
    int dsum = 0;
    int dot = 0;
    int minus = 0;
    for (int i = 0; i < len; i++)
    {
        if (isdigit(a[i]) != 0)
        {
            dsum++;
        }
        else if (a[i] == '.')
        {
            dot++;
        }
        else if (a[i] == '-')
        {
            minus++;
        }
    }
    if (dsum + dot + minus == len)
        return 1;
    else
        return 0;
}

string getting_type(char mssg[])
```

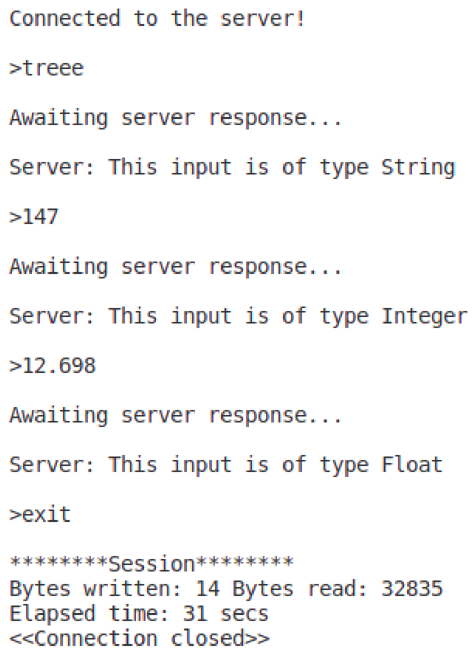
```
{
    string reply = "";
    if (isint(mssg) == 1)
    {
        reply += "This input is of type Integer";
    }
    else if (isfloat(mssg) == 1)
    {
        reply += "This input is of type Float";
    }
    else
    {
        reply += "This input is of type String";
    }
    return reply;
}

int main(int argc, char *argv[])
{
    // for the server, we only need to specify a port number
    if (argc != 2)
    {
        cerr << "Usage: port" << endl;
        exit(0);
    }
    // grab the port number
    int port = atoi(argv[1]);
    // buffer to send and receive messages with
    char msg[1500];

    // setup a socket and connection tools
    sockaddr_in servAddr;
    bzero((char *)&servAddr, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(port);

    // open stream oriented socket with internet address
    // also keep track of the socket descriptor
    int serverSd = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSd < 0)
    {
        cerr << "Error establishing the server socket" << endl;
        exit(0);
    }
    // bind the socket to its local address
    int bindStatus = bind(serverSd, (struct sockaddr *)&servAddr,
                          sizeof(servAddr));
    if (bindStatus < 0)
    {
        cerr << "Error binding socket to local address" << endl;
        exit(0);
    }
    cout << "Waiting for a client to connect..." << endl;
    // listen for up to 5 requests at a time
    listen(serverSd, 5);
    // receive a request from client using accept
    // we need a new address to connect with the client
    sockaddr_in newSockAddr;
    socklen_t newSockAddrSize = sizeof(newSockAddr);
    // accept, create a new socket descriptor to
    // handle the new connection with client
    int newSd = accept(serverSd, (sockaddr *)&newSockAddr, &newSockAddrSize);
    if (newSd < 0)
    {
        cerr << "Error accepting request from client!" << endl;
        exit(1);
    }
    cout << "Connected with client!" << endl;
    // lets keep track of the session time
    struct timeval start1, end1;
    gettimeofday(&start1, NULL);
    // also keep track of the amount of data sent as well
    int bytesRead, bytesWritten = 0;
    while (1)
    {
        // receive a message from the client (listen)
        cout << "\nAwaiting client response..." << endl;
        memset(&msg, 0, sizeof(msg)); // clear the buffer
        bytesRead += recv(newSd, (char *)&msg, sizeof(msg), 0);
        if (!strcmp(msg, "exit"))
        {
            cout << "\nClient has quit the session" << endl;
            break;
        }
        string msg_type = getting_type(msg);
        // cout << "Server says this is a : " << msg_type << "\n";
        cout << "\nClient: " << msg << endl;
        string data;
        // getline(cin, data);
        // memset(&msg, 0, sizeof(msg)); // clear the buffer
        strcpy(msg, msg_type.c_str());
        // send the message to client
        bytesWritten += send(newSd, (char *)&msg, strlen(msg), 0);
        // bytesWritten += send(newSd, (string *)&msg_type, msg_type.size(), 0);

        // send to the client that server has closed the connection
        // string exx = "exit";
        // strcpy(msg, exx.c_str());
        // send(newSd, (char *)&msg, strlen(msg), 0);
    }
    // we need to close the socket descriptors after we're all done
    gettimeofday(&end1, NULL);
    close(newSd);
    close(serverSd);
    cout << "\n********Session********" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "Connection closed..." << endl;
    return 0;
}

---------|| client.cpp ||-----------
#include <iostream>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;
// Client side
int main(int argc, char *argv[])
{
    // we need 2 things: ip address and port number, in that order
    if (argc != 3)
    {
        cerr << "Usage: ip_address port" << endl;
        exit(0);
    } // grab the IP address and port number
    char *serverIp = argv[1];
    int port = atoi(argv[2]);
    // create a message buffer
    char msg[1500];
    // setup a socket and connection tools
    struct hostent *host = gethostbyname(serverIp);
    sockaddr_in sendSockAddr;
    bzero((char *)&sendSockAddr, sizeof(sendSockAddr));
    sendSockAddr.sin_family = AF_INET;
    sendSockAddr.sin_addr.s_addr =
        inet_addr(inet_ntoa(*(struct in_addr *)*host->h_addr_list));
    sendSockAddr.sin_port = htons(port);
    int clientSd = socket(AF_INET, SOCK_STREAM, 0);
    // try to connect...
    int status = connect(clientSd, (sockaddr *)&sendSockAddr, sizeof(sendSockAddr));
    if (status < 0)
    {
        cout << "Error connecting to socket!" << endl;
        exit(EXIT_FAILURE);
    }
    cout << "Connected to the server!" << endl;
    int bytesRead, bytesWritten = 0;
    struct timeval start1, end1;
    string choice;
    gettimeofday(&start1, NULL);
    while (1)
    {
        cout << "\n>";
        string data;
        getline(cin, data);
        memset(&msg, 0, sizeof(msg)); // clear the buffer
        strcpy(msg, data.c_str());
        if (data == "exit")
        {
            send(clientSd, (char *)&msg, strlen(msg), 0);
            break;
        }
        bytesWritten += send(clientSd, (char *)&msg, strlen(msg), 0);
        cout << "\nAwaiting server response..." << endl;
        memset(&msg, 0, sizeof(msg)); // clear the buffer
        bytesRead += recv(clientSd, (char *)&msg, sizeof(msg), 0);
        cout << "\nServer: " << msg << endl;
    }
    gettimeofday(&end1, NULL);
    close(clientSd);
    cout << "\n********Session********" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "<<Connection closed>>" << endl;
    return 0;
}
```

```
___________Output___________
Server :                                Client :
```



```
Waiting for a client to connect...
Connected with client!

Awaiting client response...

Client: treee

Awaiting client response...

Client: 147

Awaiting client response...

Client: 12.698

Awaiting client response...

Client has quit the session

********Session********
Bytes written: 84 Bytes read: -1463307353
Elapsed time: 31 secs
Connection closed...
```

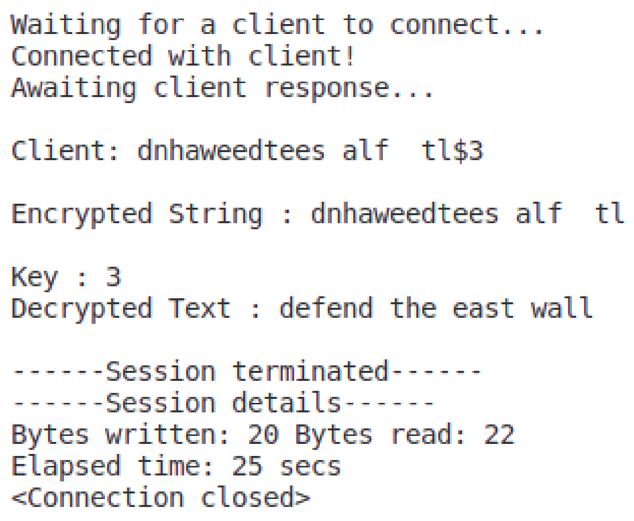


```
Connected to the server!

>treee

Awaiting server response...

Server: This input is of type String

>147

Awaiting server response...

Server: This input is of type Integer

>12.698

Awaiting server response...

Server: This input is of type Float

>exit

********Session********
Bytes written: 14 Bytes read: 32835
Elapsed time: 31 secs
<<Connection closed>>
```

```
------------------------------ Program 8 ---------------------------------
----------------- Server-Client with Rail-Fencing Cipher ------------------

---------|| server.cpp ||-----------
// Server side
#include <iostream>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;

string EncryptRail(string text, int key)
{
    char rail[key][(text.length())];
    for (int i = 0; i < key; i++)
        for (int j = 0; j < text.length(); j++)
            rail[i][j] = '\n';
    bool dir_down = false;
    int row = 0, col = 0;
    for (int i = 0; i < text.length(); i++)
    {
        if (row == 0 || row == key - 1)
            dir_down = !dir_down;
        rail[row][col++] = text[i];
        dir_down ? row++ : row--;
    }
    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;
}

string DecryptRail(string cipher, int key)
{
    char rail[key][cipher.length()];
    for (int i = 0; i < key; i++)
        for (int j = 0; j < cipher.length(); j++)
            rail[i][j] = '\n';
    bool dir_down;
    int row = 0, col = 0;
    for (int i = 0; i < cipher.length(); i++)
    {
        if (row == 0)
            dir_down = true;
        if (row == key - 1)
            dir_down = false;
        rail[row][col++] = '*';
        dir_down ? row++ : row--;
    }
    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++];
    string result;
    row = 0, col = 0;
    for (int i = 0; i < cipher.length(); i++)
    {
        if (row == 0)
            dir_down = true;
        if (row == key - 1)
            dir_down = false;
        if (rail[row][col] != '*')
            result.push_back(rail[row][col++]);
        dir_down ? row++ : row--;
    }
    return result;
}

// function to decrypt
string decrypt(string cipher_text, string key)
{
    string orig_text;

    for (int i = 0; i < cipher_text.size(); i++)
    {
        // converting in range 0-25
        char x = (cipher_text[i] - key[i] + 26) % 26;

        // convert into alphabets(ASCII)
        x += 'A';
        orig_text.push_back(x);
    }
    return orig_text;
}
string generateKey(string str, string key)
{
    int x = str.size();

    for (int i = 0;; i++)
    {
        if (x == i)
            i = 0;
        if (key.size() == str.size())
            break;
        key.push_back(key[i]);
    }
    return key;
}
string parsing_str(string mssg)
{
    string a = "";
    for (int i = 0; i < mssg.size(); i++)
    {
        if (mssg[i] == '$')
        {
            break;
        }
        else
            a += mssg[i];
    }
    return a;
}

string parsing_key(string mssg)
{
    string a = "";
    int idx;
    for (int i = 0; i < mssg.size(); i++)
    {
        if (mssg[i] == '$')
        {
            idx = i;
            break;
        }
        // else a+=mssg[i];
    }
    a = mssg.substr(idx + 1, mssg.size() - (idx + 1));
    return a;
}

// main function
int main(int argc, char *argv[])
{
    // for the server, we only need to specify a port number
    if (argc != 2)
    {
        cerr << "Usage: port" << endl;
        exit(0);
    }
    // grab the port number
    int port = atoi(argv[1]);

    // setup a socket and connection tools
    sockaddr_in servAddr;
    bzero((char *)&servAddr, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(port);

    // open stream oriented socket with internet address
    // also keep track of the socket descriptor
    int serverSd = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSd < 0)
    {
        cerr << "Error establishing the server socket" << endl;
        exit(0);
    }
    // binding the socket to its local address
    int bindStatus = bind(serverSd, (struct sockaddr *)&servAddr,
                          sizeof(servAddr));
    if (bindStatus < 0)
    {
        cerr << "Error binding socket to local address" << endl;
        exit(0);
    }
    cout << "Waiting for a client to connect..." << endl;
    // listen for up to 5 requests at a time
    listen(serverSd, 5);

    // receive a request from client using accept
    // we need a new address to connect with the client
    sockaddr_in newSockAddr;
    socklen_t newSockAddrSize = sizeof(newSockAddr);
    // accept, create a new socket descriptor to
    // handle the new connection with client
    int newSd = accept(serverSd, (sockaddr *)&newSockAddr, &newSockAddrSize);
    if (newSd < 0)
    {
        cerr << "Error accepting request from client!" << endl;
        exit(1);
    }
    cout << "Connected with client!" << endl;

    // keeping track of the session time and amount of data sent
    struct timeval start1, end1;
    gettimeofday(&start1, NULL);
    int bytesRead, bytesWritten = 0;

    // buffer to send and receive messages with
    char msg[1500];
    // variables
    string received_string;
    bool checker;
    string data;

    cout << "Awaiting client response..." << endl;
    memset(&msg, 0, sizeof(msg)); // clear the buffer
    bytesRead += recv(newSd, (char *)&msg, sizeof(msg), 0);
    received_string = msg;
    cout << "\nClient: " << received_string << endl;
    // differentiating btwn key and string
    string str = parsing_str(received_string);
    string key = parsing_key(received_string);
    cout << "\nEncrypted String : " << str << "\n";

    // key = generateKey(str, key);
    cout << "\nKey : " << key << "\n";
    int k = stoi(key);

    // decrypt
    data = DecryptRail(str, k);

    memset(&msg, 0, sizeof(msg)); // clear the buffer
    cout << "Decrypted Text : " << data << endl;
    // send the message to client
    strcpy(msg, data.c_str());
    bytesWritten += send(newSd, (char *)&msg, strlen(msg), 0);

    // closing the socket descriptors
    gettimeofday(&end1, NULL);
    close(newSd);
    close(serverSd);
    cout << "\n------Session terminated------" << endl;
    cout << "------Session details------" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "<Connection closed>" << endl;
    return 0;
}

---------|| client.cpp ||-----------
// Client side
#include <bits/stdc++.h>
#include <iostream>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;

string EncryptRail(string text, int key)
{
    char rail[key][(text.length())];
    for (int i = 0; i < key; i++)
        for (int j = 0; j < text.length(); j++)
            rail[i][j] = '\n';
    bool dir_down = false;
    int row = 0, col = 0;
    for (int i = 0; i < text.length(); i++)
    {
        if (row == 0 || row == key - 1)
            dir_down = !dir_down;
        rail[row][col++] = text[i];
        dir_down ? row++ : row--;
    }
    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;
}

string generateKey(string str, string key)
{
    int x = str.size();

    for (int i = 0;; i++)
    {
        if (x == i)
            i = 0;
        if (key.size() == str.size())
            break;
        key.push_back(key[i]);
    }
    return key;
}

string encrypt(string str, string key)
{
    string cipher_text;

    for (int i = 0; i < str.size(); i++)
    {
        // converting in range 0-25
        char x = (str[i] + key[i]) % 26;

        // convert into alphabets(ASCII)
        x += 'A';

        cipher_text.push_back(x);
    }
    return cipher_text;
}

int main(int argc, char *argv[])
{
    //  two things req: ip address and port number respectively
    if (argc != 3)
    {
        cerr << "Usage: ip_address port" << endl;
        exit(0);
    }

    // grab the IP address and port number
    char *serverIp = argv[1];
    int port = atoi(argv[2]);

    // setup a socket and connection tools
    struct hostent *host = gethostbyname(serverIp);
    sockaddr_in sendSockAddr;
    bzero((char *)&sendSockAddr, sizeof(sendSockAddr));
    sendSockAddr.sin_family = AF_INET;
    sendSockAddr.sin_addr.s_addr = inet_addr(inet_ntoa(*(struct in_addr *)*host-
>h_addr_list));
    sendSockAddr.sin_port = htons(port);
    int clientSd = socket(AF_INET, SOCK_STREAM, 0);
    // try to connect...
    int status = connect(clientSd, (sockaddr *)&sendSockAddr, sizeof(sendSockAddr));
    if (status < 0)
    {
        cout << "Error connecting to socket!" << endl;
        exit(EXIT_FAILURE);
    }
    cout << "Connected to the server!" << endl;

    // keeping track of the session time and amount of data sent
    int bytesRead, bytesWritten = 0;
    struct timeval start1, end1;
    gettimeofday(&start1, NULL);

    // creating a message buffer
    char msg[1500];
    string data, key;

    cout << "Enter data string" << endl;
    cout << ">>>";
    // taking input string from client
    getline(cin, data);
    cout << "Enter key" << endl;
    cout << ">>>";
    // taking input key from client
    getline(cin, key);
    // toupper(data);
    transform(data.begin(), data.end(), data.begin(), ::tolower);
    transform(key.begin(), key.end(), key.begin(), ::tolower);
    // key = toupper(key);
    int keyword = stoi(key);
    //
    // key = generateKey(data, key);

    // calling cipher function
    string ans = EncryptRail(data, keyword);
    memset(&msg, 0, sizeof(msg)); // clear the buffer
    cout << "<< Sending Encrypted Data : " << ans << " >>\n";

    ans = ans + '$' + key;

    // sending data to the server
    strcpy(msg, ans.c_str());
    bytesWritten += send(clientSd, (char *)&msg, strlen(msg), 0);

    cout << "Awaiting server response..." << endl;
    memset(&msg, 0, sizeof(msg)); // clear the buffer

    // receiving server's message
    bytesRead += recv(clientSd, (char *)&msg, sizeof(msg), 0);
    cout << "\nServer: " << msg << endl;

    // closing the socket descriptors
    gettimeofday(&end1, NULL);
    close(clientSd);
    cout << "\n------Session terminated------" << endl;
    cout << "------Session details------" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "<Connection closed>" << endl;
    return 0;
}
```

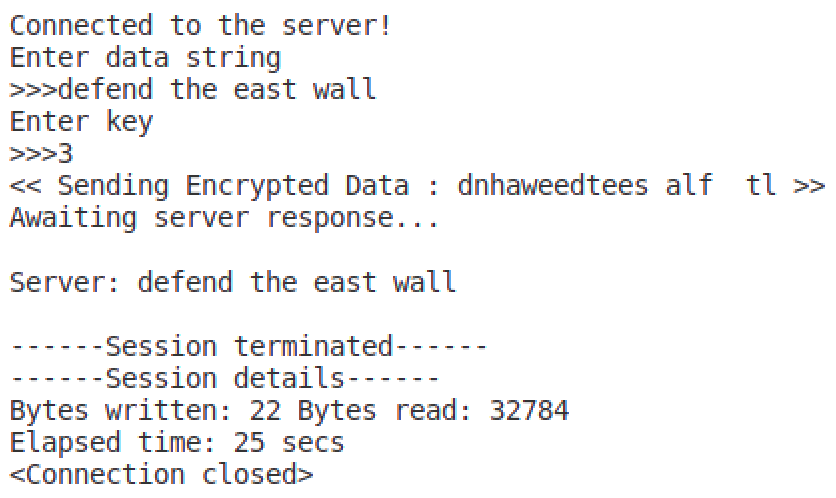
___________Output___________

Server :



```
Waiting for a client to connect...
Connected with client!
Awaiting client response...

Client: dnhaweedtees alf  tl$3

Encrypted String : dnhaweedtees alf  tl

Key : 3
Decrypted Text : defend the east wall

------Session terminated------
------Session details------
Bytes written: 20 Bytes read: 22
Elapsed time: 25 secs
<Connection closed>
```

Client :



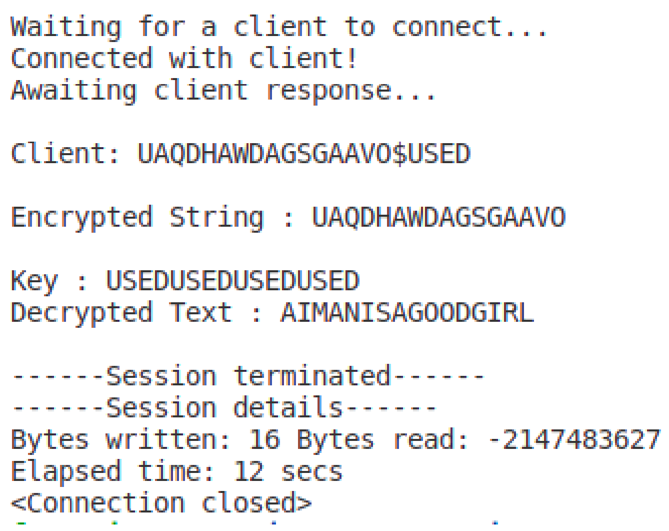
```
Connected to the server!
Enter data string
>>>defend the east wall
Enter key
>>>3
<< Sending Encrypted Data : dnhaweedtees alf  tl >>
Awaiting server response...

Server: defend the east wall

------Session terminated------
------Session details------
Bytes written: 22 Bytes read: 32784
Elapsed time: 25 secs
<Connection closed>
```

```
------------------------------- Program 9 ----------------------------------
-------------------- Server-Client with Vigenere Cipher ---------------------

---------|| server.cpp ||-----------
// Server side
#include <iostream>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;

// function to decrypt
string decrypt(string cipher_text, string key)
{
    string orig_text;

    for (int i = 0; i < cipher_text.size(); i++)
    {
        // converting in range 0-25
        char x = (cipher_text[i] - key[i] + 26) % 26;

        // convert into alphabets(ASCII)
        x += 'A';
        orig_text.push_back(x);
    }
    return orig_text;
}
string generateKey(string str, string key)
{
    int x = str.size();

    for (int i = 0;; i++)
    {
        if (x == i)
            i = 0;
        if (key.size() == str.size())
            break;
        key.push_back(key[i]);
    }
    return key;
}
string parsing_str(string mssg)
{
    string a = "";
    for (int i = 0; i < mssg.size(); i++)
    {
        if (mssg[i] == '$')
        {
            break;
        }
        else
            a += mssg[i];
    }
    return a;
}

string parsing_key(string mssg)
{
    string a = "";
    int idx;
    for (int i = 0; i < mssg.size(); i++)
    {
        if (mssg[i] == '$')
        {
            idx = i;
            break;
        }
        // else a+=mssg[i];
    }
    a = mssg.substr(idx + 1, mssg.size() - (idx + 1));
    return a;
}

// main function
int main(int argc, char *argv[])
{
    // for the server, we only need to specify a port number
    if (argc != 2)
    {
        cerr << "Usage: port" << endl;
        exit(0);
    }
    // grab the port number
    int port = atoi(argv[1]);

    // setup a socket and connection tools
    sockaddr_in servAddr;
    bzero((char *)&servAddr, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(port);

    // open stream oriented socket with internet address
    // also keep track of the socket descriptor
    int serverSd = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSd < 0)
    {
        cerr << "Error establishing the server socket" << endl;
        exit(0);
    }
    // binding the socket to its local address
    int bindStatus = bind(serverSd, (struct sockaddr *)&servAddr,
                          sizeof(servAddr));
    if (bindStatus < 0)
    {
        cerr << "Error binding socket to local address" << endl;
        exit(0);
    }
    cout << "Waiting for a client to connect..." << endl;
    // listen for up to 5 requests at a time
    listen(serverSd, 5);

    // receive a request from client using accept
    // we need a new address to connect with the client
    sockaddr_in newSockAddr;
    socklen_t newSockAddrSize = sizeof(newSockAddr);
    // accept, create a new socket descriptor to
    // handle the new connection with client
    int newSd = accept(serverSd, (sockaddr *)&newSockAddr, &newSockAddrSize);
    if (newSd < 0)
    {
        cerr << "Error accepting request from client!" << endl;
        exit(1);
    }
    cout << "Connected with client!" << endl;

    // keeping track of the session time and amount of data sent
    struct timeval start1, end1;
    gettimeofday(&start1, NULL);
    int bytesRead, bytesWritten = 0;

    // buffer to send and receive messages with
    char msg[1500];
    // variables
    string received_string;
    bool checker;
    string data;

    cout << "Awaiting client response..." << endl;
    memset(&msg, 0, sizeof(msg)); // clear the buffer
    bytesRead += recv(newSd, (char *)&msg, sizeof(msg), 0);
    received_string = msg;
```

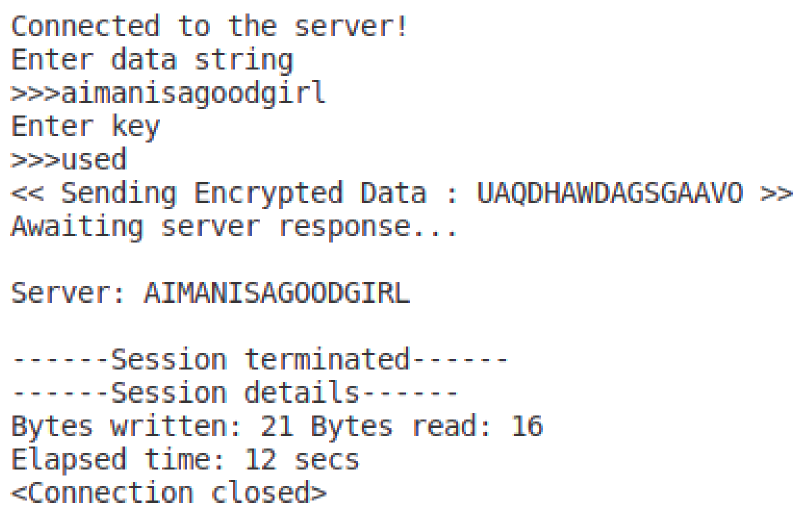
```
    cout << "\nClient: " << received_string << endl;

    // differentiating btwn key and string
    string str = parsing_str(received_string);
    string key = parsing_key(received_string);
    cout << "\nEncrypted String : " << str << "\n";

    key = generateKey(str, key);
    cout << "\nKey : " << key << "\n";

    // decrypt
    data = decrypt(str, key);

    memset(&msg, 0, sizeof(msg)); // clear the buffer
    cout << "Decrypted Text : " << data << endl;
    // send the message to client
    strcpy(msg, data.c_str());
    bytesWritten += send(newSd, (char *)&msg, strlen(msg), 0);

    // closing the socket descriptors
    gettimeofday(&end1, NULL);
    close(newSd);
    close(serverSd);
    cout << "\n------Session terminated------" << endl;
    cout << "------Session details------" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "<Connection closed>" << endl;
    return 0;
}

---------|| client.cpp ||-----------
// Client side
#include <bits/stdc++.h>
#include <iostream>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <fstream>
using namespace std;

string generateKey(string str, string key)
{
    int x = str.size();

    for (int i = 0;; i++)
    {
        if (x == i)
            i = 0;
        if (key.size() == str.size())
```

```
            break;
        key.push_back(key[i]);
    }
    return key;
}

string encrypt(string str, string key)
{
    string cipher_text;

    for (int i = 0; i < str.size(); i++)
    {
        // converting in range 0-25
        char x = (str[i] + key[i]) % 26;

        // convert into alphabets(ASCII)
        x += 'A';

        cipher_text.push_back(x);
    }
    return cipher_text;
}

int main(int argc, char *argv[])
{
    //  two things req: ip address and port number respectively
    if (argc != 3)
    {
        cerr << "Usage: ip_address port" << endl;
        exit(0);
    }

    // grab the IP address and port number
    char *serverIp = argv[1];
    int port = atoi(argv[2]);

    // setup a socket and connection tools
    struct hostent *host = gethostbyname(serverIp);
    sockaddr_in sendSockAddr;
    bzero((char *)&sendSockAddr, sizeof(sendSockAddr));
    sendSockAddr.sin_family = AF_INET;
    sendSockAddr.sin_addr.s_addr = inet_addr(inet_ntoa(*(struct in_addr *)*host-
>h_addr_list));
    sendSockAddr.sin_port = htons(port);
    int clientSd = socket(AF_INET, SOCK_STREAM, 0);
    // try to connect...
    int status = connect(clientSd, (sockaddr *)&sendSockAddr, sizeof(sendSockAddr));
    if (status < 0)
    {
        cout << "Error connecting to socket!" << endl;
        exit(EXIT_FAILURE);
    }
    cout << "Connected to the server!" << endl;

    // keeping track of the session time and amount of data sent
    int bytesRead, bytesWritten = 0;
    struct timeval start1, end1;
    gettimeofday(&start1, NULL);

    // creating a message buffer
    char msg[1500];
    string data, key, keyword;
    cout << "Enter data string" << endl;
    cout << ">>>";
    // taking input string from client
    getline(cin, data);
    cout << "Enter key" << endl;
    cout << ">>>";
    // taking input key from client
    getline(cin, key);
    // toupper(data);
    transform(data.begin(), data.end(), data.begin(), ::toupper);
    transform(key.begin(), key.end(), key.begin(), ::toupper);
    // key = toupper(key);
    keyword = key;
    //
    key = generateKey(data, key);

    // calling cipher function
    string ans = encrypt(data, key);
    memset(&msg, 0, sizeof(msg)); // clear the buffer
    cout << "<< Sending Encrypted Data : " << ans << " >>\n";

    ans = ans + '$' + keyword;

    // sending data to the server
    strcpy(msg, ans.c_str());
    bytesWritten += send(clientSd, (char *)&msg, strlen(msg), 0);

    cout << "Awaiting server response..." << endl;
    memset(&msg, 0, sizeof(msg)); // clear the buffer

    // receiving server's message
    bytesRead += recv(clientSd, (char *)&msg, sizeof(msg), 0);
    cout << "\nServer: " << msg << endl;

    // closing the socket descriptors
    gettimeofday(&end1, NULL);
    close(clientSd);
    cout << "\n------Session terminated------" << endl;
    cout << "------Session details------" << endl;
    cout << "Bytes written: " << bytesWritten << " Bytes read: " << bytesRead << endl;
    cout << "Elapsed time: " << (end1.tv_sec - start1.tv_sec)
         << " secs" << endl;
    cout << "<Connection closed>" << endl;
    return 0;
}
```

```
__________Output__________
Server :                                        Client :
```



```
Waiting for a client to connect...
Connected with client!
Awaiting client response...

Client: UAQDHAWDAGSGAAVO$USED

Encrypted String : UAQDHAWDAGSGAAVO

Key : USEDUSEDUSEDUSED
Decrypted Text : AIMANISAGOODGIRL

------Session terminated------
------Session details------
Bytes written: 16 Bytes read: -2147483627
Elapsed time: 12 secs
<Connection closed>
```



```
Connected to the server!
Enter data string
>>>aimanisagoodgirl
Enter key
>>>used
<< Sending Encrypted Data : UAQDHAWDAGSGAAVO >>
Awaiting server response...

Server: AIMANISAGOODGIRL

------Session terminated------
------Session details------
Bytes written: 21 Bytes read: 16
Elapsed time: 12 secs
<Connection closed>
```

```
------------------------------- Program 10 ----------------------------------
---------------------------- PlayFair Cipher  ------------------------------
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Playfair
{

public:
    char keyMatrix[5][5];
    string getKey()
    {
        string k;
        cout << "Enter the key : ";
        cin >> k;
        return k;
    }
    string getMessage()
    {
        string m;
        cout << "Enter the message : ";
        cin >> m;
        return m;
    }
    void encrypt(string msg, string key)
    {
        makeKeyMatrix(key);
        cout << "\n----------Encryption Process-----------\n"
             << endl;
        string bg = getBigrams(msg);
        int n = bg.length();
        vector<char> ct(n);
        int x = 0;
        while (x < n)
        {
            int a_row = 0, a_col = 0, b_row = 0, b_col = 0;
            char ca = ' ', cb = ' ';
            getPosition(bg[x], bg[x + 1], a_row, a_col, b_row, b_col);
            if (a_row == b_row)
            {
                ca = keyMatrix[a_row][mod((a_col + 1), 5)];
                cb = keyMatrix[b_row][mod((b_col + 1), 5)];
            }
            else if (a_col == b_col)
            {
                ca = keyMatrix[mod((a_row + 1), 5)][a_col];
                cb = keyMatrix[mod((b_row + 1), 5)][b_col];
            }
            else
            {
                ca = keyMatrix[a_row][b_col];
                cb = keyMatrix[b_row][a_col];
            }

            cout << bg[x] << bg[x + 1] << "--->" << ca << cb << endl;
            ct[x] = ca;
            ct[x + 1] = cb;
            x += 2;
        }
        cout << "\n>>The cipher text is :  ";
        for (int i = 0; i < n; i++)
        {
            cout << ct[i];
        }
        cout << endl;
    }

    // function for decryption
    void decrypt(string msg, string key)
    {
        makeKeyMatrix(key);
        cout << "\n----------Decryption Process------------" << endl;
        // string bg = getBigrams(msg);
        int n = msg.length();
        vector<char> pt(n);
        int x = 0;
        while (x < n)
        {
            int a_row = 0, a_col = 0, b_row = 0, b_col = 0;
            char pa = ' ', pb = ' ';
            char c1 = toupper(msg[x]);
            char c2 = toupper(msg[x + 1]);
            getPosition(c1, c2, a_row, a_col, b_row, b_col);
            // cout<<a_row<<","<<a_col<<","<<b_row<<","<<b_col;
            if (a_row == b_row)
            {
                pa = keyMatrix[a_row][mod((a_col - 1), 5)];
                pb = keyMatrix[b_row][mod((b_col - 1), 5)];
            }
            else if (a_col == b_col)
            {
                pa = keyMatrix[mod((a_row - 1), 5)][a_col];
                pb = keyMatrix[mod((b_row - 1), 5)][b_col];
            }
            else
            {
                pa = keyMatrix[a_row][b_col];
                pb = keyMatrix[b_row][a_col];
            }
```

```
            cout << msg[x] << msg[x + 1] << "--->" << pa << pb << endl;
            pt[x] = pa;
            pt[x + 1] = pb;
            x += 2;
        }
        cout << "\n>>The plain text is :  ";
        for (int i = 0; i < n; i++)
        {
            cout << pt[i];
        }
        cout << endl;
    }

private:
    string getBigrams(string msg)
    {
        int len = msg.length();
        vector<char> a(len);
        int n = 0;
        string bigrams(len, ' ');
        for (int i = 0; i < len; i++)
        {
            if (msg[i] == ' ')
            { // ignore space
                continue;
            }
            else if (msg[i] == 'j' || msg[i] == 'J')
            {
                a[n] = 'I'; // replace j or J  with I
                n++;
            }
            else
            {
                a[n] = toupper(msg[i]);
                n++;
            }
        }
        int k = 0;
        char filler = 'X';
        bool filled = false;
        for (int i = 0; i < n; i += 2)
        {
            bigrams[k] = a[i];
            k++;
            if (i + 1 != n)
            {
                if (a[i] == a[i + 1] && filled == false)
                {
                    bigrams[k] = filler;
                    k++;
                    bigrams[k] = a[i + 1];
                    k++;
                    filled = true;
                    continue;
                }
                else
                {
                    bigrams[k] = a[i + 1];
                    k++;
                }
            }
        }
        if (k % 2 != 0)
        {
            bigrams[k] = filler;
            bigrams[k + 1] = '\0';
        }
        else
        {
            bigrams[k] = '\0';
        }
        return bigrams;
    }

    // function to generate key matrix
    void makeKeyMatrix(string text)
    {
        int len = text.length();
        vector<char> ca(len);
        int n = 0;
        for (int i = 0; i < len; i++)
        {
            if (text[i] == ' ')
            {
                continue;
            }
            ca[n] = toupper(text[i]);
            n++;
        }

        // array of alphabets
        char alphabets[26];
        for (int i = 0; i < 26; i++)
        {
            alphabets[i] = i + 65;
        }

        char oneD[26];
        int p = 0;
        oneD[0] = ca[0];
        for (int i = 1; i < n; i++)
        {
            bool duplicate = false;
            for (int j = 0; j < i; j++)
            {
                if (ca[i] == ca[j])
                { // avoid duplicates
                    duplicate = true;
                    break;
                }
            }
            if (duplicate == false)
            {
                oneD[++p] = ca[i];
            }
        }
        if (p < 26)
        {
            for (int i = 0; i < 26; i++)
            {
                bool duplicate = false;
                for (int j = 0; j < p; j++)
                {
                    if (alphabets[i] == oneD[j])
                    {
                        duplicate = true;
                        break;
                    }
                }
                if (duplicate == false)
                {
                    oneD[++p] = alphabets[i];
                }
            }
            p = 0;
            for (int i = 0; i < 5; i++)
            {
                for (int j = 0; j < 5; j++)
                {
                    if (oneD[p] == 'J')
                    {
                        p++;
                    }
                    keyMatrix[i][j] = oneD[p];
                    p++;
                }
            }
            cout << "\n|| Key matrix for Playfair Cipher for given key ||\n";

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

                    cout << keyMatrix[i][j] << " ";
                }
                cout << "\n";
            }
        }
    }

    void getPosition(char a, char b, int &a_row, int &a_column, int &b_row, int &b_column)
    {
        bool match_a = false;
        bool match_b = false;
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                if (keyMatrix[i][j] == a && match_a != true)
                {
                    a_row = i;
                    a_column = j;
                    // cout<<keyMatrix[i][j];
                    match_a = true;
                }
                if (keyMatrix[i][j] == b && match_b != true)
                {
                    b_row = i;
                    b_column = j;
                    // cout<<keyMatrix[i][j];
                    match_b = true;
                }
            }
            if (match_a && match_b)
            {
                break;
            }
        }
    }
    int mod(int a, int b)
    { // to handle modulo operation
        int c = a % b;
        if (c >= 0)
            return c;
        else
            return b + c;
    }
};
int main()
{
    cout << "\n____Aiman Fatima 20BCS008___\n";
    cout << "\n----Playfair Cipher----\n";
    int choice;
    Playfair p;
    string m, k;
    while (1)
    {
        cout << "\n_____MENU____\n";
        cout << "1. Encrypt your text\n";
        cout << "2. Decrypt some encrypted text\n";
        cout << "3. Exit\n";
        cout << "\nEnter your choice : ";
        cin >> choice;
        cout << endl;
        switch (choice)
        {
        case 1:
            k = p.getKey();
            m = p.getMessage();
            p.encrypt(m, k);
            break;
        case 2:
            k = p.getKey();
            m = p.getMessage();
            p.decrypt(m, k);
            break;
        case 3:
            cout << "The End.\n\n";
            return 0;
        default:
            cout << "Wrong Choice!!\n";
        }
    }
    return 0;
}
```

____________Output____________

```
____Aiman Fatima 20BCS008___

----Playfair Cipher----

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 1

Enter the key : monarch
Enter the message : thisnotreallybad

|| Key matrix for Playfair Cipher for given key ||
M O N A R
C H B D E
F G I K L
P Q S T U
V W X Y Z

----------Encryption Process------------

TH--->QD
IS--->SX
NO--->AN
TR--->UA
EA--->DR
LX--->IZ
LY--->KZ
BA--->DN

>>The cipher text is :  QDSXANUADRIZKZDN

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 2

Enter the key : monarch
Enter the message : qdsxanuadrizkkzdn
Enter your choice : 2

Enter the key : monarch
Enter the message : qdsxanuadrizkkzdn

|| Key matrix for Playfair Cipher for given key ||
M O N A R
C H B D E
F G I K L
P Q S T U
V W X Y Z

----------Decryption Process------------
qd--->TH
sx--->IS
an--->NO
ua--->TR
dr--->EA
iz--->LX
kk--->II
zd--->YE
n--->OR

>>The plain text is :  THISNOTREALXIIYEO

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 4

Wrong Choice!!

_____MENU____
1. Encrypt your text
2. Decrypt some encrypted text
3. Exit

Enter your choice : 3

The End.
```