```
#include <winsock2.h>
#include <WS2tcpip.h>
#include <bits/stdc++.h>

// need link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

using namespace std;

int main()
{
   WORD wVersion = MAKEWORD(2, 2); // Specify the version of Winsock you want to use
                                   // 2.2 -> x=2, y=2
   WSADATA wsaData;
   if (WSAStartup(wVersion, &wsaData) != 0) // Phiên bản Winsock cần tải, Con trỏ trỏ đến cấu trúc LPWSADATA
   {
      cout << "WSAStartup failed with error" << GetLastError() << endl;
      return 1;
   }
   cout << "WSAStartup completed." << endl;

   char ip[] = "127.0.0.";
   char stCodeString[10];
   unsigned int stCode = 106200228;
   unsigned int num = (stCode % 255) + 1;
   sprintf(stCodeString, "%d", num);
   strcat(ip, stCodeString);
   cout << "IP address: " << ip << endl;

   // inet_addr
   unsigned long ip_addr1 = inet_addr(ip);
   if (ip_addr1 == INADDR_NONE)
   {
      printf("Can't convert IP by inet_addr\n");
   }
   else
   {
      cout << "Convert inet_addr completed:" << ip_addr1 << endl;
   }

   // inet_pton
   unsigned long ip_addr2;
   if (inet_pton(AF_INET, ip, &ip_addr2) == 1)
   {
      cout << "Convert inet_pton completed:" << ip_addr2 << endl;
   }
   else
   {
      cout << "Can't convert IP by inet_pton\n";
   }

   // Call WSACleanup when application finishes
   if (WSACleanup() != 0)
      cout << "Clean failed with error." << GetLastError() << endl;

   return 0;
}
```

Bài 1:

BT: CHUYỂN ĐỔI ĐỊA CHỈ IP

- **Viết 1 đoạn chương trình chuyển đổi địa chỉ IP:**
  - Dùng hàm ***inet_addr*** với tham số đầu vào:
    - cp: "127.0.0.x", với x = ([mã SV] modulo 255) + 1
  - Dùng hàm ***inet_pton*** với tham số đầu vào:
    - family: AF_INET
    - ipstr: "127.0.0.x", với x = ([mã SV] modulo 255) + 1
  - Xuất ra giá trị số (kiểu unsigned long) biểu diễn địa chỉ IP dạng nhị phân cho 2 cách trên

Bài 2: TCP: xử lý chuỗi

Server

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <bits/stdc++.h>

using namespace std;

#define SERVER_PORT 5500
#define SERVER_ADDR "127.0.0.1"
#define BUFF_MAXSIZE 1024

#pragma comment(lib, "Ws2_32.lib")

string process(string input)
{
    string output = "";
    for (char c : input)
        if (isalpha(c))
            output += tolower(c);
    return (output == "exit" || output == "quit") ? "good bye" : output;
}

int main()
{
    WORD wVersion = MAKEWORD(2, 2);
    WSADATA wsaData;

    if (WSAStartup(wVersion, &wsaData) != 0)
    {
        cout << "WSAStartup failed with error " << GetLastError() << endl;
        return 1;
    }
    cout << "WSAStartup completed." << endl;

    SOCKET listenSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (listenSock == INVALID_SOCKET)
    {
        cout << "Creating socket failed with code " << WSAGetLastError() << endl;
        WSACleanup(); // Clean up before returning
        return 1;
    }
    cout << "Creating socket completed successfully.\n";

    sockaddr_in tcpServerAddr;
    tcpServerAddr.sin_family = AF_INET;
    tcpServerAddr.sin_port = htons(SERVER_PORT);
    tcpServerAddr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

    if (bind(listenSock, (sockaddr *)&tcpServerAddr, sizeof(tcpServerAddr)) ==
SOCKET_ERROR)
    {
        cout << "Bind API failed with code " << WSAGetLastError() << endl;
        closesocket(listenSock); // Clean up before returning
        WSACleanup();
        return 1;
    }
    cout << "Bind API completed successfully.\n";

    if (listen(listenSock, 5) == SOCKET_ERROR)
    {
```

Client

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>

using namespace std;

#define SERVER_PORT 5500
#define SERVER_ADDR "127.0.0.1"

int main()
{
    WORD wVersionRequested = MAKEWORD(2,
2);
    WSADATA wSadata;
    if (WSAStartup(wVersionRequested,
&wSadata) != 0)
    {
        cout << "WSAStartup failed: \n";
        return 0;
    }
    cout << "WSAStartup completed." << endl;

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(SERVER_PORT);
    serverAddr.sin_addr.S_un.S_addr =
inet_addr(SERVER_ADDR);

    SOCKET client = socket(AF_INET,
SOCK_STREAM, IPPROTO_TCP);
    if (client == INVALID_SOCKET)
    {
        cout << "Creating socket failed with code " <<
WSAGetLastError() << endl;
        WSACleanup(); // Clean up before returning
        return 1;
    }
    cout << "Creating socket completed
successfully.\n";

    if (connect(client, (sockaddr *)&serverAddr,
sizeof(serverAddr)) == SOCKET_ERROR)
    {
        cout << "Connection failed with code " <<
WSAGetLastError() << endl;
        return 0;
    }
    cout << "Connection completed successfully." <<
endl;

    // STEP 3: Send data (example)
    const char *message = "eXit2a@";
    int bytesSent = send(client, message,
strlen(message), 0);
    if (bytesSent == SOCKET_ERROR)
    {
        cout << "Send failed with error: " <<
WSAGetLastError() << endl;
    }
```

```
        cout << "Listen failed with code " << WSAGetLastError() << endl;
        closesocket(listenSock); // Clean up before returning
        WSACleanup();
        return 1;
    }
    cout << "Server is listening for requests..." << endl;

    sockaddr_in clientAddr;
    char buff[BUFF_MAXSIZE], clientIP[INET_ADDRSTRLEN];
    int ret, clientAddrLen = sizeof(clientAddr), clientPort;

    SOCKET NewConnection = accept(listenSock, (sockaddr *)&clientAddr,
&clientAddrLen);
    if (NewConnection == INVALID_SOCKET)
    {
        cout << "Connection failed with code: " << WSAGetLastError() << endl;
        closesocket(listenSock); // Clean up before returning
        WSACleanup();
        return 1;
    }
    inet_ntop(AF_INET, &clientAddr.sin_addr, clientIP, sizeof(clientIP));
    clientPort = ntohs(clientAddr.sin_port);
    cout << "Connection is established: IP = " << clientIP << " at port = " <<
clientPort << endl;

    while (1)
    {
        ret = recv(NewConnection, buff, BUFF_MAXSIZE, 0);
        if (ret == SOCKET_ERROR)
        {
            cout << "Error with code " << WSAGetLastError() << endl;
            break;
        }
        else if (ret == 0)
        {
            cout << "Client disconnected." << endl;
            break;
        }
        else
        {
            buff[ret] = '\0'; // Null-terminate the received data
            cout << "Received message from client " << clientIP << ":" << clientPort
<< ": " << buff << endl;
            string buffStr(buff);
            buffStr = process(buffStr);
            const char *message = buffStr.c_str();
            ret = send(NewConnection, message, strlen(message), 0);
            if (ret == SOCKET_ERROR)
            {
                cout << "Error with code " << WSAGetLastError() << endl;
                break;
            }
        }
    }
    shutdown(NewConnection, SD_BOTH);

    closesocket(NewConnection);
    closesocket(listenSock);
    WSACleanup();
    return 0;
}
```

```
    // STEP 4: Receive data (example)
    char buffer[1024];
    int bytesReceived = recv(client, buffer,
sizeof(buffer), 0);
    if (bytesReceived == SOCKET_ERROR)
    {
        cout << "Receive failed with error: " <<
WSAGetLastError() << endl;
    }
    else
    {
        buffer[bytesReceived] = '\0';
        cout << "Received: " << buffer << endl;
    }

    // STEP 5: Shutdown the connection (if needed)
    shutdown(client, SD_BOTH); // SD_BOTH closes
both send and receive

    // STEP 6: Close sockets and clean up
    closesocket(client);
    WSACleanup();

    return 0;
}
```

Bài 3: UDP – chuyển DNS thành IP

Server

```
// cd UDP
// g++ UDPServer.cpp -o UDPServer.exe -lws2_32
// .\UDPServer.exe 8080

#include <bits/stdc++.h>
#include <winsock2.h> // Include the Windows Sockets API header
#include <stdio.h>
#include <ws2tcpip.h>

using namespace std;

string process(char *hostname)
{
    string st = "";
    struct addrinfo hints, *result, *rp;
    ZeroMemory(&hints, sizeof(hints));
    // hints.ai_family = AF_UNSPEC;     // Allow IPv4 or IPv6
    hints.ai_family = AF_INET;       // Allow IPv4
    hints.ai_socktype = SOCK_STREAM; // Use TCP

    int status = getaddrinfo(hostname, NULL, &hints, &result);
    if (status != 0)
    {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(status));
        WSACleanup();
        return "Not found information";
    }

    for (rp = result; rp != NULL; rp = rp->ai_next)
    {
        void *addr;
        char ipstr[INET6_ADDRSTRLEN];

        if (rp->ai_family == AF_INET)
        {
            struct sockaddr_in *ipv4 = (struct sockaddr_in *)rp->ai_addr;
            addr = &(ipv4->sin_addr);
        }
        else
        {
            struct sockaddr_in6 *ipv6 = (struct sockaddr_in6 *)rp->ai_addr;
            addr = &(ipv6->sin6_addr);
        }

        inet_ntop(rp->ai_family, addr, ipstr, sizeof(ipstr));
        string s(ipstr);
        st += "\n" + s;
    }

    freeaddrinfo(result);
    return st;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
```

Client

```
// g++ UDPclient.cpp -o UDPclient.exe -lws2_32
// .\UDPclient.exe 127.0.0.1 8080 quora.com

#include <iostream>
#include <winsock2.h>

using namespace std;

int main(int argc, char *argv[])
{
    if (argc != 4)
    {
        cerr << "Usage: clientUDP.exe <server_ip>
<port_number> <domain_name>" << endl;
        return 1;
    }

    const char *serverIP = argv[1];
    int port = atoi(argv[2]);
    const char *domainName = argv[3];

    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) !=
0)
    {
        cerr << "WSAStartup failed!" << endl;
        return 1;
    }
    cout << "WSAStartup successfully!" << endl;

    SOCKET clientSocket = socket(AF_INET,
SOCK_DGRAM, 0);
    if (clientSocket == INVALID_SOCKET)
    {
        cerr << "Socket creation failed!" << endl;
        WSACleanup();
        return 1;
    }
    cout << "Socket creates successfully!" << endl;

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(port);
    serverAddr.sin_addr.s_addr = inet_addr(serverIP);

    if (sendto(clientSocket, domainName,
strlen(domainName), 0, (sockaddr *)&serverAddr,
sizeof(serverAddr)) == SOCKET_ERROR)
    {
        cerr << "Error in sendto()" << endl;
        closesocket(clientSocket);
        WSACleanup();
        return 1;
    }

    char buffer[1024];
    sockaddr_in serverResponse;
```

```
        cerr << "Usage: serverUDP.exe <port_number>" << endl;
        return 1;
    }

    int port = atoi(argv[1]);

    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        cerr << "WSAStartup failed!" << endl;
        return 1;
    }
    cout << "WSAStartup successfully!" << endl;

    SOCKET serverSocket = socket(AF_INET, SOCK_DGRAM, 0);
    if (serverSocket == INVALID_SOCKET)
    {
        cerr << "Socket creation failed!" << endl;
        WSACleanup();
        return 1;
    }
    cout << "Socket creates successfully!" << endl;

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(port);

    if (bind(serverSocket, (sockaddr *)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR)
    {
        cerr << "Binding failed!" << endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }
    cout << "Binding successfully!" << endl;

    char buffer[1024];
    sockaddr_in clientAddr;
    int clientAddrLen = sizeof(clientAddr);

    while (true)
    {
        int bytesReceived = recvfrom(serverSocket, buffer, sizeof(buffer), 0,
(sockaddr *)&clientAddr, &clientAddrLen);
        if (bytesReceived == SOCKET_ERROR)
        {
            cerr << "Error in recvfrom()" << endl;
            break;
        }

        buffer[bytesReceived] = '\0';

        string st = process(buffer);
        const char *message = st.c_str();

        sendto(serverSocket, message, strlen(message), 0, (sockaddr *)&clientAddr,
clientAddrLen);
    }

    closesocket(serverSocket);
    int serverResponseLen = sizeof(serverResponse);

    int bytesReceived = recvfrom(clientSocket, buffer,
sizeof(buffer), 0, (sockaddr *)&serverResponse,
&serverResponseLen);
    if (bytesReceived == SOCKET_ERROR)
    {
        cerr << "Error in recvfrom()" << endl;
    }
    else
    {
        buffer[bytesReceived] = '\0';
        cout << "Resolved IP Address: " << buffer << endl;
    }

    closesocket(clientSocket);
    WSACleanup();

    return 0;
}
```

```
    WSACleanup();

    return 0;
}
```

Bài 4: Login TCP

Server

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <bits/stdc++.h>

using namespace std;

#define SERVER_PORT 5500
#define SERVER_ADDR "127.0.0.1"
#define BUFF_MAXSIZE 1024

#pragma comment(lib, "Ws2_32.lib")

struct UserData
{
    string id;
    string password;
    bool active;
    bool login;
};
bool isLogin = false;
vector<UserData> userData;
string saveUserLogged;

vector<UserData> readUserCSV(string filename)
{
    vector<UserData> userDataRead;
    ifstream file(filename);

    if (!file.is_open())
        cerr << "Error: Unable to open file " << filename << endl;

    string line;

    // Skip the header line
    getline(file, line);

    while (getline(file, line))
    {
        stringstream lineStream(line);
        string cell;
        UserData user;

        // Read id, password, active, and login
        if (getline(lineStream, user.id, ',') &&
            getline(lineStream, user.password, ',') &&
            getline(lineStream, cell, ',') &&
            stringstream(cell) >> user.active &&
            getline(lineStream, cell, ',') &&
            stringstream(cell) >> user.login)
        {
            userDataRead.push_back(user);
        }
        else
        {
```

Client

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>

using namespace std;

#define SERVER_PORT 5500
#define SERVER_ADDR "127.0.0.1"

int main()
{
    WORD wVersionRequested = MAKEWORD(2,
2);
    WSADATA wSadata;
    if (WSAStartup(wVersionRequested,
&wSadata) != 0)
    {
        cout << "WSAStartup failed: \n";
        return 0;
    }
    cout << "WSAStartup completed." << endl;

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(SERVER_PORT);
    serverAddr.sin_addr.S_un.S_addr =
inet_addr(SERVER_ADDR);

    SOCKET client = socket(AF_INET,
SOCK_STREAM, IPPROTO_TCP);
    if (client == INVALID_SOCKET)
    {
        cout << "Creating socket failed with code " <<
WSAGetLastError() << endl;
        WSACleanup(); // Clean up before returning
        return 1;
    }
    cout << "Creating socket completed
successfully.\n";

    if (connect(client, (sockaddr *)&serverAddr,
sizeof(serverAddr)) == SOCKET_ERROR)
    {
        cout << "Connection failed with code " <<
WSAGetLastError() << endl;
        return 0;
    }
    cout << "Connection completed successfully."
<< endl;

    // STEP 3: Send data (example)
    string messageSend;
    cout << "To Login: login <id> <password>\n To
Logout: logout\n";
```

```
            cerr << "Error: Invalid data format in line" << endl;
        }
    }
    file.close();
    return userDataRead;
}

void updateUserCSV(string filename)
{
    ofstream file(filename, ios::out | ios::trunc);
    if (!file.is_open())
    {
        cerr << "Error: Unable to create/open file " << filename << endl;
        return;
    }
    file << "ID,Password,Active,Login\n";
    for (const UserData &user : userData)
    {
        file << user.id << "," << user.password << "," << user.active << "," <<
user.login << "\n";
    }
    file.close();
}

string process(const string &input)
{
    userData = readUserCSV("database.csv"); // Update data from database
    istringstream iss(input);
    string request, id, password, temp;
    iss >> request >> id >> password >> temp;

    if (request == "login" && id != "" && password != "" && temp == "")
    {
        if (isLogin == 1)
            return "You logged in";
        for (UserData &user : userData)
        {
            if (user.id == id)
            {
                if (user.password == password)
                {
                    isLogin = 1;
                    saveUserLogged = user.id;
                    user.login = 1;
                    updateUserCSV("database.csv");
                    return "Login successfully";
                }
                return "Wrong password";
            }
        }
        return "Id does not exist";
    }
    else if (request == "logout" && id == "") // Check if redundant params
    {
        if (isLogin == 0)
            return "Please login";
        for (auto &user : userData)
            if (user.id == saveUserLogged)
                user.login = 0;
        isLogin = false;
        updateUserCSV("database.csv");
        return "Logout successfully";
```

```
    while (1)
    {
        cout << "(Client) -->";
        getline(cin, messageSend);
        const char *message = messageSend.c_str();
        int bytesSent = send(client, message,
strlen(message), 0);
        if (bytesSent == SOCKET_ERROR)
        {
            cout << "Send failed with error: " <<
WSAGetLastError() << endl;
        }

        // STEP 4: Receive data (example)
        char buffer[1024];
        int bytesReceived = recv(client, buffer,
sizeof(buffer), 0);
        if (bytesReceived == SOCKET_ERROR)
        {
            cout << "Receive failed with error: " <<
WSAGetLastError() << endl;
        }
        else
        {
            buffer[bytesReceived] = '\0';
            cout << "(Server) " << buffer << endl;
        }
    }

    // STEP 5: Shutdown the connection (if needed)
    shutdown(client, SD_BOTH); // SD_BOTH
closes both send and receive

    // STEP 6: Close sockets and clean up
    closesocket(client);
    WSACleanup();
    return 0;
}
```

```
    }
    else
        return "Wrong message format";
}
int main()
{
    WORD wVersion = MAKEWORD(2, 2);
    WSADATA wsaData;

    if (WSAStartup(wVersion, &wsaData) != 0)
    {
        cout << "WSAStartup failed with error " << GetLastError() << endl;
        return 1;
    }
    cout << "WSAStartup completed." << endl;

    SOCKET listenSock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (listenSock == INVALID_SOCKET)
    {
        cout << "Creating socket failed with code " << WSAGetLastError() << endl;
        WSACleanup(); // Clean up before returning
        return 1;
    }
    cout << "Creating socket completed successfully.\n";

    sockaddr_in tcpServerAddr;
    tcpServerAddr.sin_family = AF_INET;
    tcpServerAddr.sin_port = htons(SERVER_PORT);
    tcpServerAddr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

    if (bind(listenSock, (sockaddr *)&tcpServerAddr, sizeof(tcpServerAddr)) ==
SOCKET_ERROR)
    {
        cout << "Bind API failed with code " << WSAGetLastError() << endl;
        closesocket(listenSock); // Clean up before returning
        WSACleanup();
        return 1;
    }
    cout << "Bind API completed successfully.\n";

    if (listen(listenSock, 5) == SOCKET_ERROR)
    {
        cout << "Listen failed with code " << WSAGetLastError() << endl;
        closesocket(listenSock); // Clean up before returning
        WSACleanup();
        return 1;
    }
    cout << "Server is listening for requests..." << endl;

    sockaddr_in clientAddr;
    char buff[BUFF_MAXSIZE], clientIP[INET_ADDRSTRLEN];
    int ret, clientAddrLen = sizeof(clientAddr), clientPort;

    SOCKET NewConnection = accept(listenSock, (sockaddr *)&clientAddr,
&clientAddrLen);
    if (NewConnection == INVALID_SOCKET)
    {
        cout << "Connection failed with code: " << WSAGetLastError() << endl;
        closesocket(listenSock); // Clean up before returning
        WSACleanup();
        return 1;
    }
```

```
    inet_ntop(AF_INET, &clientAddr.sin_addr, clientIP, sizeof(clientIP));
    clientPort = ntohs(clientAddr.sin_port);
    cout << "Connection is established: IP = " << clientIP << " at port = " <<
clientPort << endl;

    while (1)
    {
        ret = recv(NewConnection, buff, BUFF_MAXSIZE, 0);
        if (ret == SOCKET_ERROR)
        {
            cout << "Error with code " << WSAGetLastError() << endl;
            break;
        }
        else if (ret == 0)
        {
            cout << "Client disconnected." << endl;
            break;
        }
        else
        {
            buff[ret] = '\0'; // Null-terminate the received data
            cout << "Received message from client " << clientIP << ":" << clientPort
<< ": " << buff << endl;
            string buffStr(buff);
            buffStr = process(buffStr);
            const char *message = buffStr.c_str();
            ret = send(NewConnection, message, strlen(message), 0);
            if (ret == SOCKET_ERROR)
            {
                cout << "Error with code " << WSAGetLastError() << endl;
                break;
            }
        }
    }

    shutdown(NewConnection, SD_BOTH);

    closesocket(NewConnection);
    closesocket(listenSock);
    WSACleanup();
    return 0;
}
```

Bài 5: Login UDP

Server

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <bits/stdc++.h>

using namespace std;

#define SERVER_PORT 5500
#define SERVER_ADDR "127.0.0.1"
#define BUFF_MAXSIZE 1024

#pragma comment(lib, "Ws2_32.lib")

struct UserData
{
    string id;
    string password;
```

Client

```
// g++ UDPclient.cpp -o UDPclient.exe -lws2_32
// .\UDPclient.exe 127.0.0.1 8080 dut.udn.vn

#include <iostream>
#include <winsock2.h>

using namespace std;

#define SERVER_PORT 5500
#define SERVER_ADDR "127.0.0.1"
int main()
{
    const char *serverIP = SERVER_ADDR;
    int port = SERVER_PORT;

    WSADATA wsaData;
```

```
    bool active;
    bool login;
};
bool isLogin = false;
vector<UserData> userData;
string saveUserLogged;

vector<UserData> readUserCSV(string filename)
{
    vector<UserData> userDataRead;
    ifstream file(filename);

    if (!file.is_open())
        cerr << "Error: Unable to open file " << filename << endl;

    string line;

    // Skip the header line
    getline(file, line);

    while (getline(file, line))
    {
        stringstream lineStream(line);
        string cell;
        UserData user;

        // Read id, password, active, and login
        if (getline(lineStream, user.id, ',') &&
            getline(lineStream, user.password, ',') &&
            getline(lineStream, cell, ',') &&
            stringstream(cell) >> user.active &&
            getline(lineStream, cell, ',') &&
            stringstream(cell) >> user.login)
        {
            userDataRead.push_back(user);
        }
        else
        {
            cerr << "Error: Invalid data format in line" << endl;
        }
    }
    file.close();
    return userDataRead;
}

void updateUserCSV(string filename)
{
    ofstream file(filename, ios::out | ios::trunc);
    if (!file.is_open())
    {
        cerr << "Error: Unable to create/open file " << filename << endl;
        return;
    }
    file << "ID,Password,Active,Login\n";
    for (const UserData &user : userData)
    {
        file << user.id << "," << user.password << "," << user.active << ","
<< user.login << "\n";
    }
    file.close();
}
```

```
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        cerr << "WSAStartup failed!" << endl;
        return 1;
    }
    cout << "WSAStartup completed." << endl;

    // Create socket
    SOCKET clientSocket = socket(AF_INET, SOCK_DGRAM,
0);
    if (clientSocket == INVALID_SOCKET)
    {
        cerr << "Socket creation failed!" << endl;
        WSACleanup();
        return 1;
    }
    cout << "Creating socket completed successfully.\n";

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(port);
    serverAddr.sin_addr.s_addr = inet_addr(serverIP);

    string messageSend;
    while (1)
    {
        cout << "(Client) -->";
        getline(cin, messageSend);
        const char *message = messageSend.c_str();
        if (sendto(clientSocket, message, strlen(message), 0,
(sockaddr *)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR)
        {
            cerr << "Error in sendto()" << endl;
            closesocket(clientSocket);
            WSACleanup();
            return 1;
        }

        char buffer[1024];
        sockaddr_in serverResponse;
        int serverResponseLen = sizeof(serverResponse);

        int bytesReceived = recvfrom(clientSocket, buffer,
sizeof(buffer), 0, (sockaddr *)&serverResponse,
&serverResponseLen);
        if (bytesReceived == SOCKET_ERROR)
        {
            cerr << "Error in recvfrom()" << endl;
        }
        else
        {
            buffer[bytesReceived] = '\0';
            cout << "(Server): " << buffer << endl;
        }
    }
    closesocket(clientSocket);
    WSACleanup();

    return 0;
}
```

```
string process(const string &input)
{
   userData = readUserCSV("database.csv"); // Update data from
database
   istringstream iss(input);
   string request, id, password, temp;
   iss >> request >> id >> password >> temp;

   if (request == "login" && id != "" && password != "" && temp ==
"")
   {
      if (isLogin == 1)
         return "You logged in";
      for (UserData &user : userData)
      {
         if (user.id == id)
         {
            if (user.password == password)
            {
               isLogin = 1;
               saveUserLogged = user.id;
               user.login = 1;
               updateUserCSV("database.csv");
               return "Login successfully";
            }
            return "Wrong password";
         }
      }
      return "Id does not exist";
   }
   else if (request == "logout" && id == "") // Check if redundant params
   {
      if (isLogin == 0)
         return "Please login";
      for (auto &user : userData)
         if (user.id == saveUserLogged)
            user.login = 0;
      isLogin = false;
      updateUserCSV("database.csv");
      return "Logout successfully";
   }
   else
      return "Wrong message format";
}

int main()
{
   WORD wVersion = MAKEWORD(2, 2);
   WSADATA wsaData;

   if (WSAStartup(wVersion, &wsaData) != 0)
   {
      cout << "WSAStartup failed with error " << GetLastError() <<
endl;
      return 1;
   }
   cout << "WSAStartup completed." << endl;

   SOCKET udpServerSock = socket(AF_INET, SOCK_DGRAM,
IPPROTO_UDP);

   if (udpServerSock == INVALID_SOCKET)
```

```
    {
        cout << "Creating socket failed with code " << WSAGetLastError()
<< endl;
        WSACleanup();
        return 1;
    }
    cout << "Creating socket completed successfully.\n";

    sockaddr_in udpServerAddr;
    udpServerAddr.sin_family = AF_INET;
    udpServerAddr.sin_port = htons(SERVER_PORT);
    udpServerAddr.sin_addr.s_addr = inet_addr(SERVER_ADDR);

    if (bind(udpServerSock, (sockaddr *)&udpServerAddr,
sizeof(udpServerAddr)) == SOCKET_ERROR)
    {
        cout << "Bind API failed with code " << WSAGetLastError() <<
endl;
        closesocket(udpServerSock);
        WSACleanup();
        return 1;
    }
    cout << "Bind API completed successfully.\n";

    sockaddr_in clientAddr;
    char buff[BUFF_MAXSIZE], clientIP[INET_ADDRSTRLEN];
    int ret, clientAddrLen = sizeof(clientAddr), clientPort;

    while (1)
    {
        ret = recvfrom(udpServerSock, buff, BUFF_MAXSIZE, 0,
(sockaddr *)&clientAddr, &clientAddrLen);
        if (ret == SOCKET_ERROR)
        {
            cout << "Error with code " << WSAGetLastError() << endl;
            break;
        }
        else
        {
            inet_ntop(AF_INET, &clientAddr.sin_addr, clientIP,
sizeof(clientIP));
            clientPort = ntohs(clientAddr.sin_port);
            buff[ret] = '\0';
            cout << "Received message from client " << clientIP << ":" <<
clientPort << ": " << buff << endl;
            string buffStr(buff);
            buffStr = process(buffStr);
            const char *message = buffStr.c_str();
            ret = sendto(udpServerSock, message, strlen(message), 0,
(sockaddr *)&clientAddr, clientAddrLen);
            if (ret == SOCKET_ERROR)
            {
                cout << "Error with code " << WSAGetLastError() << endl;
                break;
            }
        }
    }
    closesocket(udpServerSock);
    WSACleanup();
    return 0;
}
```