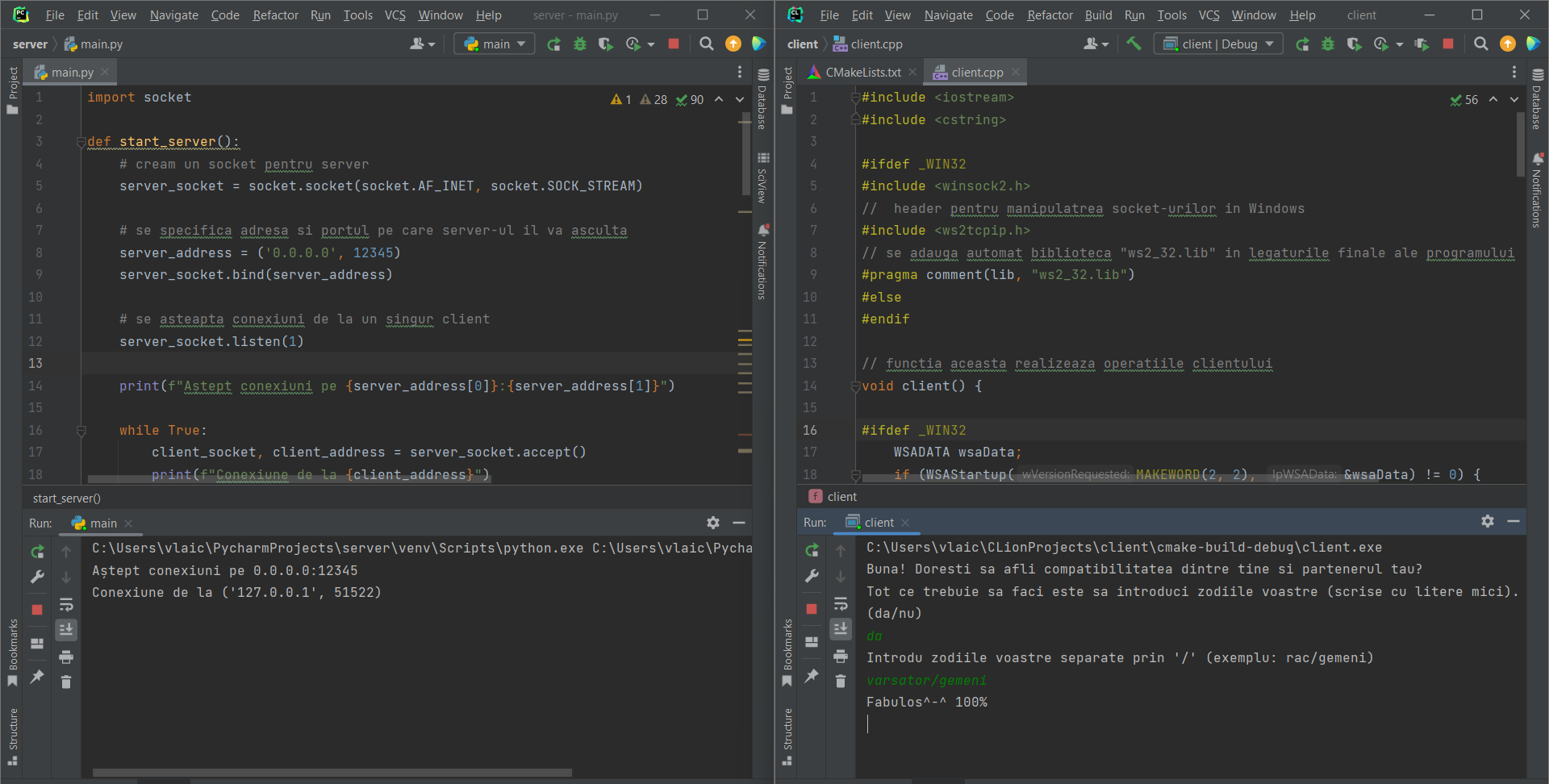
Proiect TCP/UDP

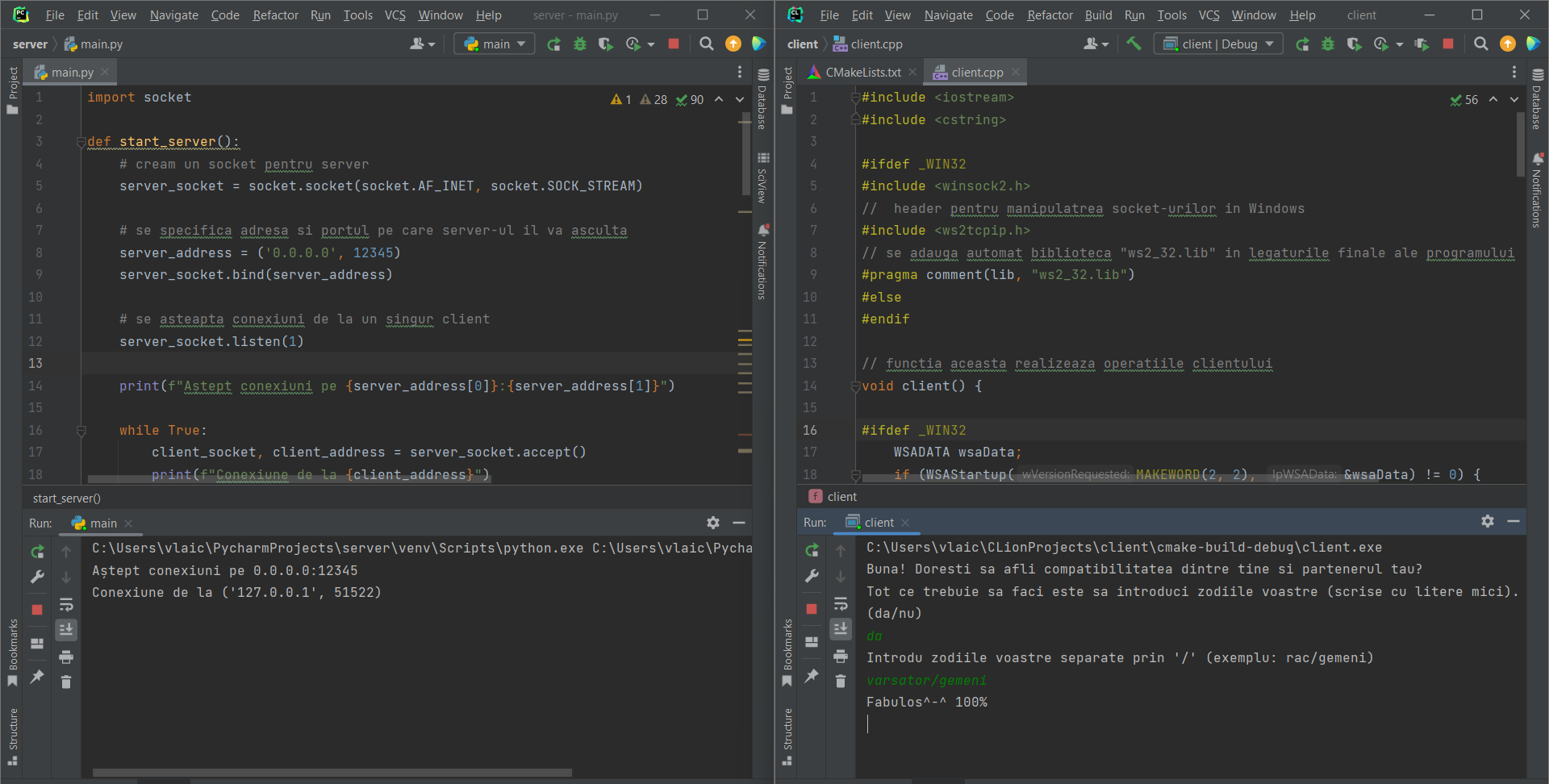
In acest proiect, prin legatura dintre client si server, clientul afla compatibilitatea dintre zodia sa si o alta zodie aleasa de el, prin intermediul server-ului care o calculeaza intr-o functie si o transmite inapoi clientului. Daca clientul nu doreste sa afle compatibilitatea sau nu introduce datele conform cerintelor precizate, server-ul trimite mesajul “o zi buna!!” . De asemenea, daca exista vreo eroare pe parcurs, se vor afisa diverse mesaje sugestive de eroare.

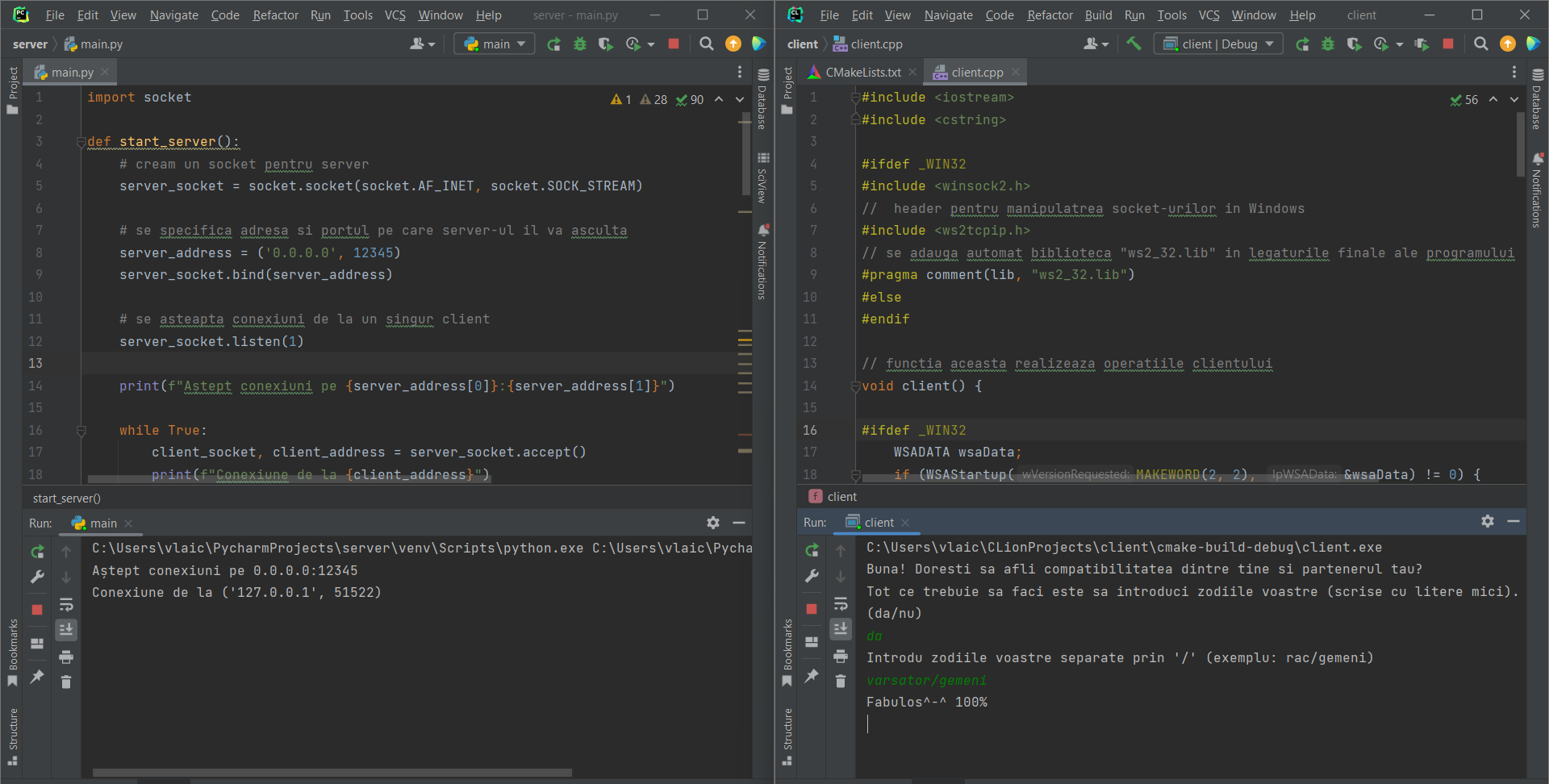
**TCP:**

**Server-ul in Python:**

import socket  
  
def start\_server():  
 # cream un socket pentru server  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 # se specifica adresa si portul pe care server-ul il va asculta  
 server\_address = ('0.0.0.0', 12345)  
 server\_socket.bind(server\_address)  
  
 # se asteapta conexiuni de la un singur client  
 server\_socket.listen(1)  
  
 print(f"Aștept conexiuni pe {server\_address[0]}:{server\_address[1]}")  
  
 while True:  
 client\_socket, client\_address = server\_socket.accept()  
 print(f"Conexiune de la {client\_address}")  
  
 # trimite mesajul initial catre client  
 compatibility\_message = "Buna! Doresti sa afli compatibilitatea dintre tine si partenerul tau? " \  
 "\nTot ce trebuie sa faci este sa introduci zodiile voastre (scrise cu litere mici).\n(da/nu)"  
 client\_socket.sendall(compatibility\_message.encode('utf-8'))  
  
 # asteapta raspunsul de la client  
 response = client\_socket.recv(1024).decode('utf-8')  
  
 if response.lower() == 'da':  
 # trimite intrebarea legata de zodii  
 client\_socket.sendall("Introdu zodiile voastre separate prin '/' (exemplu: rac/gemeni)".encode('utf-8'))  
  
 # asteapta raspunsul  
 zodiacs = client\_socket.recv(1024).decode('utf-8')  
  
 # calculeaza prin functia "calculate\_compatibility" ceea ce i s-a cerut  
 compatibility\_result = calculate\_compatibility(zodiacs)  
  
 # trimite rezultatul obtinut la client  
 client\_socket.sendall(compatibility\_result.encode('utf-8'))  
 else:  
 # daca clientul selecteaza initial "nu" atunci se va trimite acest mesaj  
 client\_socket.sendall("o zi buna!! ".encode('utf-8'))  
  
 # conexiunea cu clientul se incheie  
 client\_socket.close()  
  
def calculate\_compatibility(zodiacs):  
 # implementarea logicii functiei de calculare a compatibilitatii  
 sign1, sign2 = zodiacs.split('/');  
 if sign1 == 'varsator' and sign2 in ('gemeni' , 'balanta', 'berbec' , 'sagetator'):  
 return "Fabulos^-^ 100%"  
 elif sign1 == 'berbec' and sign2 in ( 'leu' ,'sagetator' , 'gemeni' , 'varsator'):  
 return "Se poate mai bine 75%"  
 elif sign1 == 'scorpion' and sign2 in ( 'berbec' , 'leu' , 'taur' , 'gemeni'):  
 return "Totul bine 98%"  
 elif sign1 == 'pesti' and sign2 in ( 'pesti' , 'rac' , 'scorpion' , 'taur'):  
 return "Perfect 99.9%"  
 elif sign1 == 'taur' and sign2 in ( 'varsator' , 'leu' , 'sagetator' , 'scorpion' ):  
 return "Oribil 21%"  
 elif sign1 == 'rac' and sign2 in ( 'rac' , 'scorpion' 'pesti' , 'fecioara'):  
 return "Suflete pereche 110%"  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # se initiaza server-ul atunci cand rulam  
 start\_server()  
  
  
**Clientul in C++:**  
#include <iostream>  
#include <cstring>  
  
#ifdef \_WIN32  
#include <winsock2.h>  
// header pentru manipulatrea socket-urilor in Windows  
#include <ws2tcpip.h>  
// se adauga automat biblioteca "ws2\_32.lib" in legaturile finale ale programului  
#pragma comment(lib, "ws2\_32.lib")  
#else  
#endif  
  
// functia aceasta realizeaza operatiile clientului  
void client() {  
  
#ifdef \_WIN32  
 WSADATA wsaData;  
 if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {  
 std::cerr << "Error initializing Winsock" << std::endl;  
 return;  
 }  
#endif  
 // cream socket-ul  
 int s = socket(AF\_INET, SOCK\_STREAM, 0);  
 if (s == -1) {  
 std::cerr << "am intampinat o eroare la crearea socket-ului!" << std::endl;  
#ifdef \_WIN32  
 // se elibereaza resursele Winsock in cazul in care se intampina o eroare  
 WSACleanup();  
#endif  
 return;  
 }  
  
 // se specifica adresa si portul server-ului  
 sockaddr\_in serverAddress;  
 serverAddress.sin\_family = AF\_INET;  
 serverAddress.sin\_port = htons(12345);  
 // se converteste adresa IP din text in binar  
 if (inet\_pton(AF\_INET, "127.0.0.1", &serverAddress.sin\_addr) <= 0) {  
 std::cerr << "eroare adresa ip!" << std::endl;  
#ifdef \_WIN32  
 closesocket(s);  
 WSACleanup();  
#else  
 close(s);  
#endif  
 return;  
 }  
  
 // se incearca conectarea la server  
 if (connect(s, (struct sockaddr\*)&serverAddress, sizeof(serverAddress)) == -1) {  
 std::cerr << "eroare" << std::endl;  
#ifdef \_WIN32  
 closesocket(s);  
 WSACleanup();  
#else  
 close(s);  
#endif  
 return;  
 }  
  
 char buffer[1024];  
 // se primesc/trimit date intre client si server  
 while (true) {  
 // se primeste mesajul de la server  
 ssize\_t bytesReceived = recv(s, buffer, sizeof(buffer), 0);  
 if (bytesReceived <= 0) {  
 break;  
 }  
 buffer[bytesReceived] = '\0';  
 std::cout << buffer << std::endl;  
  
 // se asteapta raspunsul de la client si se trimite la server  
 std::string raspuns;  
 std::cin >> raspuns;  
 send(s, raspuns.c\_str(), raspuns.size(), 0);  
 }  
  
  
#ifdef \_WIN32  
 closesocket(s);  
 // se elibereaza resursele Winsock dupa terminarea conversatiei  
 WSACleanup();  
#else  
 close(s);  
#endif  
}  
  
int main() {  
 // se apeleaza functia client pentru a incepe conversatia  
 client();  
 return 0;  
}







**UDP:**

**Server-ul in Python:**

import socket  
  
def start\_server():  
 # cream un socket pentru server  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
 # se specifica adresa si portul pe care server-ul il va asculta  
 server\_address = ('0.0.0.0', 12345)  
  
 print(f"Aștept conexiuni pe {server\_address[0]}:{server\_address[1]}")  
 server\_socket.bind(server\_address)  
  
 while True:  
 data, client\_address = server\_socket.recvfrom(1024)  
 print(f"Conexiune de la {client\_address}")  
  
 # trimite mesajul initial catre client  
 compatibility\_message = "Buna! Doresti sa afli compatibilitatea dintre tine si partenerul tau? " \  
 "\nTot ce trebuie sa faci este sa introduci zodiile voastre (scrise cu litere mici).\n(da/nu)"  
 server\_socket.sendto(compatibility\_message.encode('utf-8'), client\_address)  
  
 # asteapta raspunsul de la client  
 response, \_ = server\_socket.recvfrom(1024)  
 response = response.decode('utf-8')  
  
 if response.lower() == 'da':  
 # trimite intrebarea legata de zodii  
 server\_socket.sendto("Introdu zodiile voastre separate prin '/' (exemplu: rac/gemeni)".encode('utf-8'),client\_address)  
  
 # asteapta raspunsul  
 zodiacs, \_ = server\_socket.recvfrom(1024)  
 zodiacs = zodiacs.decode('utf-8')  
  
 # calculeaza prin functia "calculate\_compatibility" ceea ce i s-a cerut  
 compatibility\_result = calculate\_compatibility(zodiacs)  
  
 # trimite rezultatul obtinut la client  
 server\_socket.sendto(compatibility\_result.encode('utf-8'), client\_address)  
 else:  
 # daca clientul selecteaza initial "nu" atunci se va trimite acest mesaj  
 server\_socket.sendto("o zi buna!! ".encode('utf-8'), client\_address)  
 if response.lower() == 'exit':  
 break  
  
def calculate\_compatibility(zodiacs):  
 # implementarea logicii functiei de calculare a compatibilitatii  
 sign1, sign2 = zodiacs.split('/');  
 if sign1 == 'varsator' and sign2 in ('gemeni' , 'balanta', 'berbec' , 'sagetator'):  
 return "Fabulos^-^ 100%"  
 elif sign1 == 'berbec' and sign2 in ( 'leu' ,'sagetator' , 'gemeni' , 'varsator'):  
 return "Se poate mai bine 75%"  
 elif sign1 == 'scorpion' and sign2 in ( 'berbec' , 'leu' , 'taur' , 'gemeni'):  
 return "Totul bine 98%"  
 elif sign1 == 'pesti' and sign2 in ( 'pesti' , 'rac' , 'scorpion' , 'taur'):  
 return "Perfect 99.9%"  
 elif sign1 == 'taur' and sign2 in ( 'varsator' , 'leu' , 'sagetator' , 'scorpion' ):  
 return "Oribil 21%"  
 elif sign1 == 'rac' and sign2 in ( 'rac' , 'scorpion' 'pesti' , 'fecioara'):  
 return "Suflete pereche 110%"  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # se porneste server-ul atunci cand rulam  
 start\_server()

**Clientul in C++:**

#include <iostream>  
#include <cstring>  
  
#ifdef \_WIN32  
#include <winsock2.h>  
// header pentru manipulatrea socket-urilor in Windows  
#include <ws2tcpip.h>  
// se adauga automat biblioteca "ws2\_32.lib" in legaturile finale ale programului  
#pragma comment(lib, "ws2\_32.lib")  
#else  
#endif  
  
// functia aceasta realizeaza operatiile clientului  
void client() {  
  
#ifdef \_WIN32  
 WSADATA wsaData;  
 if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {  
 std::cerr << "Error initializing Winsock" << std::endl;  
 return;  
 }  
#endif  
 // cream socket-ul  
 int s = socket(AF\_INET, SOCK\_DGRAM, 0);  
 if (s == -1) {  
 std::cerr << "am intampinat o eroare la crearea socket-ului!" << std::endl;  
#ifdef \_WIN32  
 // se elibereaza resursele Winsock in cazul in care se intampina o eroare  
 WSACleanup();  
#endif  
 return;  
 }  
  
 // se specifica adresa si portul server-ului  
 sockaddr\_in serverAddress;  
 serverAddress.sin\_family = AF\_INET;  
 serverAddress.sin\_port = htons(12345);  
 // se converteste adresa IP din text in binar  
 if (inet\_pton(AF\_INET, "127.0.0.1", &serverAddress.sin\_addr) <= 0) {  
 std::cerr << "eroare adresa ip!" << std::endl;  
#ifdef \_WIN32  
 closesocket(s);  
 WSACleanup();  
#else  
 close(s);  
#endif  
 return;  
 }  
  
 char buffer[1024];  
 // se primesc/trimit date intre client si server  
 while (true) {  
 sendto(s, "Conectare la server", sizeof("Conectare la server"), 0, (struct sockaddr\*)&serverAddress, sizeof(serverAddress));  
 std::cout << "astept raspuns de la server" << std::endl;  
 socklen\_t serverAddressLength = sizeof(serverAddress);  
 ssize\_t bytesReceived = recvfrom(s, buffer, sizeof(buffer), 0, (struct sockaddr\*)&serverAddress, &serverAddressLength);  
 if (bytesReceived <= 0) {  
 break;  
 }  
 buffer[bytesReceived] = '\0';  
 std::cout << buffer << std::endl;  
  
 // se asteapta raspunsul de la client si se trimite la server  
 std::string raspuns;  
 std::cin >> raspuns;  
 sendto(s, raspuns.c\_str(), raspuns.size(), 0, (struct sockaddr\*)&serverAddress, sizeof(serverAddress));  
 }  
  
  
#ifdef \_WIN32  
 closesocket(s);  
 // se elibereaza resursele Winsock dupa terminarea conversatiei  
 WSACleanup();  
#else  
 close(s);  
#endif  
}  
  
int main() {  
 // se apeleaza functia client pentru a incepe conversatia  
 client();  
 return 0;  
}

