Palindrome Checker

By Yuhang Liao

Palindrome Checker is a server/client software designed for checking if the text is a palindrome or not. The palindrome is defined as a string that is equal to the reversed string. However, in this project, we just checked only numbers and letters. The cases of letters are ignored.

The project is based on socket. The client sends strings to the server. He server will check the result if it is a palindrome or not. Then, the server will send back “yes” or “no” to answer the question. The program supports multi-threaded requests; therefore, it can process more than one client at the same time.

My job is to create the entire software. I created four parts to achieve the goal. The four parts are: MyClient, MyServer, PalindromeCheckerClient and PalindromeCheckerServer. The language used in this project is C# with DotNet Core framework developed by Microsoft. DotNet Core is the framework that enables C# to support multiplatform functionality.

Basically, there is a listener in servers-side of the program. The server accepts any connection from clients and messages from clients. When a new connection requests, the server will assign a new client in server-side computer to handle the connection. The listener doesn’t handle the messages directly. It is newly assigned clients’ responsibility to send any message back to clients. The server class also checks the message if it is palindrome or not, and it sends back any result to clients if the message is not disconnection request. Both server and client are featured with multi-thread functionalities, so this program supports multiple connections from more than one clients, and process the data from them simultaneously. Unfortunately, the built-in API of DotNet Core doesn’t support the idea, so I have to make my own classes.

Firstly, I want to introduce MyClient, there are:

private readonly TcpClient Client; // TcpClient from built-in API.

private readonly Thread ClientThread; // The thread listening to any incoming message.

private bool IsRunning = false; // The status of the thread.

These are properties the client has. Its purpose is to simplify the TcpClient that is built-in API of DotNet Core. ClientThread makes the client be able to listen to the incoming message constantly. IsRunning indicates the current status of the client. If it is false, the client will be closed and disposed.

There is a delegate in this class:

public delegate void NewMessageDelegate(object sender, byte[] data);

public event NewMessageDelegate NewMessageEvent;

When new message comes to the client, it will trigger the event. The event is a callback function, so we can write custom actions after we get the message from the server.

There are two constructors in this class. One is for initializing the client with default option, another one is for initializing the client when the server accepts a new connection request.

There are three basic methods:

public void Connect(IPAddress ip, int port);

public void Send(byte[] data);

public void SendString(string data)

The first one is simply to let client connect the sever with server’s IP address and port number provided. A thread that handles incoming messages will be immediately started after the successful connection. The second one is to send the data in binary format. The third one is to send the data in string format. However, it will still translate the string format into binary, it is just a simplified version of sending method.

There are two destructors:

public void Dispose();

~MyClient();

Both are for freeing any resources the class uses in the program.

The second class is the server. Although the name is “MyListener”, it is a listener that accepts users’ connection requests.

The properties of this class are:

private readonly TcpListener Listener;

private readonly List<MyClient> Clients = new List<MyClient>();

private readonly Thread ListenerThread;

private bool IsRunning = false;

First is the built-in listener. The purpose of this class is also a simplification of built-in API. The second is a list for all clients connecting to this server. The third is the thread handling new connection requests from clients. The forth indicates the status of the server. Server will be closed immediately when the value is false.

There are two delegates in this class:

public delegate void NewMessageDelegate(object sender, object client, byte[] data);

public event NewMessageDelegate NewMessageEvent = null;

public delegate void NewClientDelegate(object sender, object newClient);

public event NewClientDelegate NewClientEvent = null;

The first delegate handles any incoming connection requests and store all clients in the list. It will also do a callback function to let other programmers to do their custom actions when the event triggered.

The second delegate handles any incoming message from clients. It will resend the message to other custom handler developed by other programmers.

There is only one constructor:

public MyListener(int port)

Users should provide a port number to initialize the server. The thread handling new connection requests will also be immediately initialized. The constructor will also initialize a callback function called:

private void NewClient\_NewMessageEvent(object sender, byte[] data)

This method will trigger the second handler mentioned above. Therefore, other programmers can do some custom functions for this event.

There is one method:

public void Start();

It starts the server and the thread.

There are two destructors:

public void Dispose();

~MyListener();

It frees any resource of this class after it is closed.

Therefore, there are server and client classes. I will introduce the client class first.

Client class is an independent program, so it is a class containing a Main method. The parameters of the main method will tell the program what parameters users provide for this client. Usually, the parameters include both IP address and port number of servers.

Firstly, it tries to convert users’ input. If it is failed, the program will be immediately closed. Then, the client will try to connect the server. If the connection is successful, a new callback function will be initialized:

private static void Client\_NewMessageEvent(object sender, byte[] data);

The method will be triggered when new message from server arrives. Sender is who triggers the event, and the data is the actual message that the server sends. This function will convert the binary data into a string and print it on the console. The string is the answer for user’s input that server sends.

There is a while loop that constantly gets the input from users. The client will send any string that the users provide to the client. And it will print any result back from server, when the callback method mentioned above is triggered. If the client gets a null or empty string from users, it will send a disconnection request and close the program.

Finally, it is the server of the Palindrome Checker. The server is also an independent program, so there is a main method. The main method uses port number as the parameter to initialize the server. The server has two callback functions:

private static void Server\_NewClientEvent(object sender, object newClient);

private static void Server\_NewMessageEvent(object sender, object client, byte[] data);

The first one is triggered when new connection request arrives. Sender means which server triggers the event and newClient means which client sends the request.

The second method is triggered when new message comes. This method also checks the message if it is a palindrome and send back the result to the client requesting this server. Sender means which server triggers the event, client means which client sends the message and the data is the message that the client sends.

Before the core algorithm checking if the string is palindrome, the server will firstly check if the data is an empty or null string. If it is, the server will decrease the number of connected clients by 1, because this message means disconnection request. When the number is zero, the server will be automatically closed.

The algorithm of checking the palindrome is simple. Firstly, the string is normalized, which means convert all uppercases to lowercases and only number and letters will be stored; all others will be discarded. There are two variables: l and r. L starts from the beginning of the string, and r starts from the end of the string. Then, the algorithm will be if the characters at l and at r are equal. The server will immediately send back “No” if there are not equal. The server will send “Yes” when l == r.

Back to main method, there is a while loop that enables the server to constantly listen both requests from servers and the new message. The server will repeatedly print the status of the server as lines in the console. It tells users what the status the server currently is.

To start the server, use this command:

./PalindromeCheckerServer.exe -port=1200

It provides the port number that the server should use.

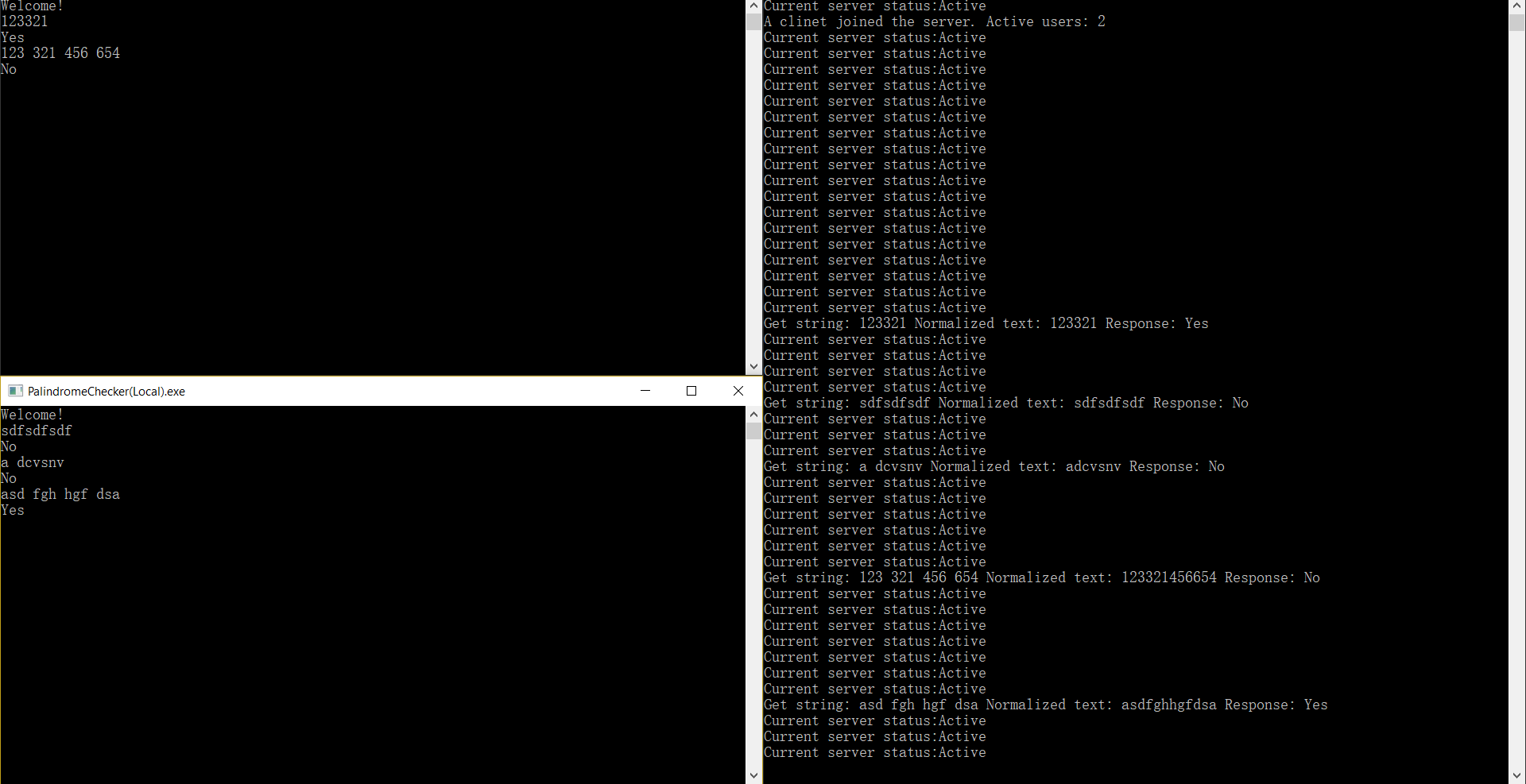
To start the client, use this command:

./PalindromeChecker.exe -ip=127.0.0.1 -port=1200

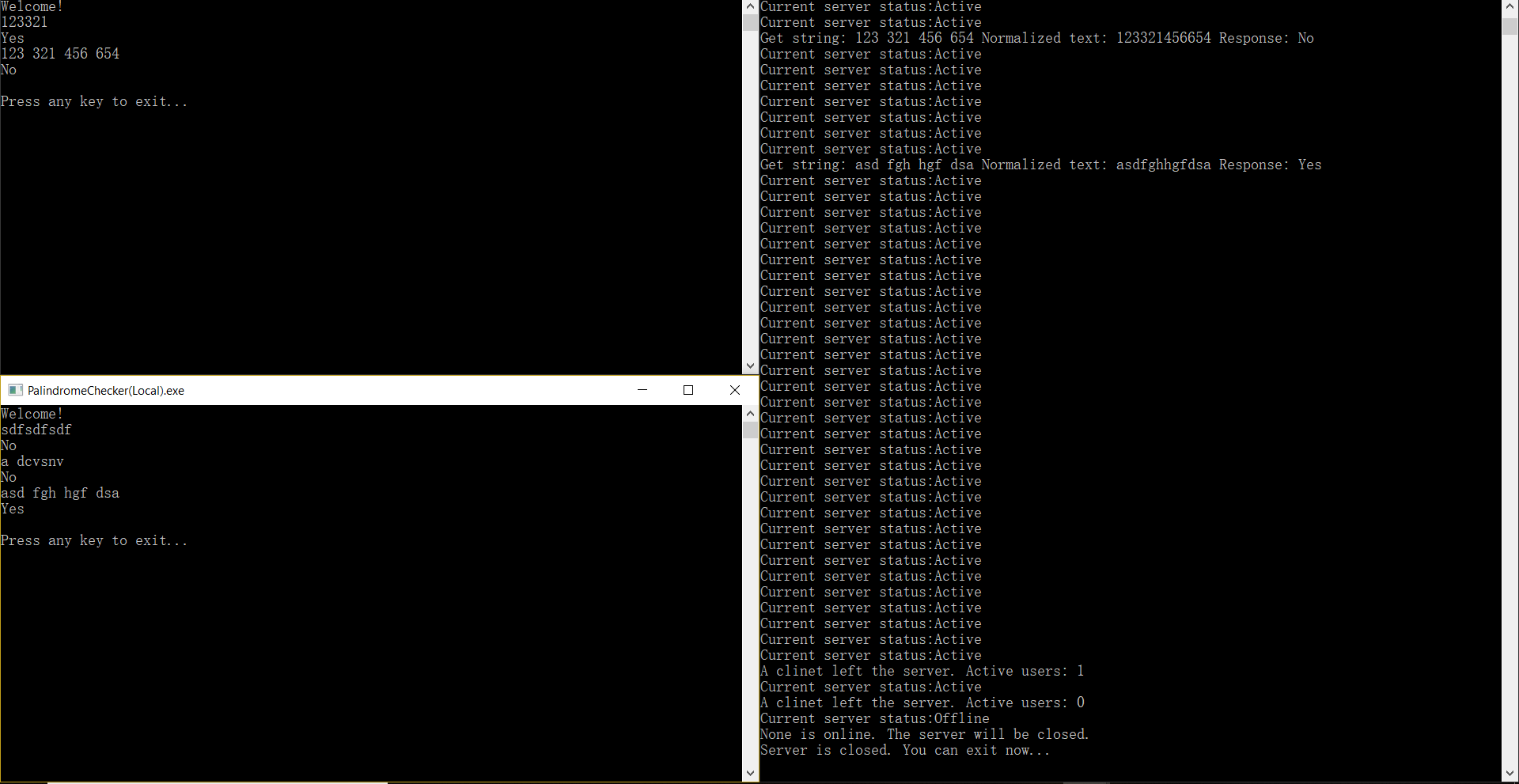
It provides the IP address and the port number where the server is.

Remember to start the server BEFORE the client!

Here are some screenshots of the result:



Disconnection requests:



This test is based on local environment using 127.0.0.1.

In conclusion, it was very easy to implement this program. The hardest parts were how to handle the connection, multithreading and the palindrome algorithm. It was a very valuable experience to me. It was also an opportunity to improve the programming skills. On the other hand, teamwork was also important in this project. Only two people working together could delivery a good result. In real development, it is still very important to separate development and testing. It is how to develop a excellent program.

GitHub: <https://github.com/yliao1/PalindromeChecker>