```
#include <iostream>
#include <string>
#include <WS2tcpip.h> //Window Socket version 2 for TCPIP
#pragma comment(lib, "ws2_32.lib")
//Compiler will print some comment to instruct linker to include library ws2_32.lib for runtime.

//Note that TCP is for reliable transmission (if packet is not received properly, it will continue to
re-send the packet).
//UDP is for performance; even if the sending is not received, it will not re-send the packet.

using namespace std;
void main() {
	string ipAddress = "127.0.0.1"; //Server's ip address; ipV4
	//const char * ipA = “127.0.0.1”;//\0
	//char const * ipA …


	//ipV6: 128 bits 2^96 times more addresses
	//ipV4: 32 bits
	//loop-back address -- it says the target ip will stay in the same computer
	//for development and testing purpose

	int port = 54000; //the listening port number on the server

	//A unique port is needed for a socket application
	//Port numbers range from 0 to 65535, but only port numbers 0 to 1023
	// are reserved for privileged services and designated as “well-known” ports.


	//Initialize Windows socket environment
	WSAData data;
//The WSADATA structure contains information about the Windows Sockets implementation.
//See: https://msdn.microsoft.com/enus/library/windows/desktop/ms741563(v=vs.85).aspx

	WORD ver = MAKEWORD(2, 2);  //See:
//http://www.cplusplus.com/forum/beginner/11110/  for WORD type definition
	//MAKEWORD(2,2) create a WORD type object with values 0x22

	int wsResult = WSAStartup(ver, &data); //Ininitlaize version 2.2 Windoes Socket
environment and store the related information in data
	if (wsResult != 0) { // value 0 will be returned if the initialization succeeds.
		cerr << "Can't start winsock, Err #" << wsResult << endl;
//if error occurs, the return value will be error number
		WSACleanup(); //Clean up Windows socket environment.
		return;
	}
	//create socket
	SOCKET sock = socket(AF_INET, SOCK_STREAM, 0); //AF: address family INET: ipV4
```

```
	//Returns a socket number. 0-65535 are valid.  -1: error
	//Create a socket.  AF_INET: ipV4;   SOCK_STREAM: TCP; 0: regular IP protocol;
                                        // AF_INET6: ipV6 ; SOCK_DGRAM: UDP
	if (sock == INVALID_SOCKET) {// -1
		cerr << "Can't create socke, Err #" << WSAGetLastError() << endl;
		// WSAGetLastError() returns the last error number
		WSACleanup();
		return;
	}

	//Bind the socket
	//Fill in a hint structure
	//See: https://msdn.microsoft.com/en-us/library/zx63b042.aspx
	sockaddr_in hint;
	//hint will have information regarding what server and what port we want to connect
	//specific socket related information for our application
	hint.sin_family = AF_INET;//ipV4
	hint.sin_port = htons(port); //54000
	//htons (host to network short) converts host short type to network short type.
	//Netowrk is big-endian; windows is little-endian
	/*
	Big Endian Byte Order: The most significant byte (the "big end") of the data is placed at
the byte with the lowest address.
	The rest of the data is placed in order in the next three bytes in memory.

	Little Endian Byte Order: The least significant byte (the "little end") of the data is placed
at the byte with the lowest address.
	The rest of the data is placed in order in the next three bytes in memory.
	*/

	inet_pton(AF_INET, ipAddress.c_str(), &hint.sin_addr); //const char *
	//inet_pton(AF_INET, "127.0.0.1", &hint.sin_addr); //this is also OK.

	//inet_pton function converts an IPv4 or IPv6 Internet network address
	// in its standard text presentation form into its network binary form.
	//ipAddress.c_str() convert C++ string into pointer to C style pointer
	// to char array (i.e., "const char *") and
	//add an addition char to ipAddress when converting it to array of char
	//pton: pointer to string to
	//ipAddress is of type string
	//network address requires  pointer to C-stype char array

	//Connect to server
	int connResult = connect(sock, (sockaddr*)&hint, sizeof(hint));
//typecasting from sockaddr_in * to sockaddr *
	if (connResult == SOCKET_ERROR) { //return 0 if no error.
		cerr << "Can't connect to server, Err #" << WSAGetLastError() << endl;
		WSACleanup();
```

```
		return;
	}

	//Send and receive
	//Do-While loop to send and receive data
	char buf[4096];
	//all TCP sockets require a buffer space on both sending and receiving sides to hold data
	string userInput;
	do {
		//prompt the user for some text
		cout << "> ";
		getline(cin, userInput); //get the whole line
		int sendResult = send(sock, userInput.c_str(), userInput.size() + 1, 0);
			//remember the size of network array of array if string size +1
		if (sendResult != SOCKET_ERROR) {
			ZeroMemory(buf, 4096);
			//put zeros to location starting at address buf, with size 4096
			int byteReceived = recv(sock, buf, 4096, 0);
			if (byteReceived > 0) {
				cout << "SERVER> " << string(buf, 0, byteReceived) << endl;
			}

		}
	} while (userInput.size() > 0);
	//Gracefully close down everything
	closesocket(sock);
	WSACleanup();
}
```