課題 01_1 server 側　01ServerAssignment.cpp 完成版

```
//-------------------------------------------------
//オンラインゲームプログラミング２　課題1_1(サーバー側)
//　　　　今回は、このサーバー側プログラムと、配布の確認用クライアント実行サンプルの組み合わせで
//　　　　実行、データのやり取りができるか確認します。
//

#include "01ServerAssignment.h"
#define NONBLOCKING_MODE 1
#define BUFFER_SIZE     1024

//-------------------------------------------------------------------------------------
// サーバ側修了確認スレッド
void ServerAssignment01::Exit()
{
        while (loop) {
                std::string input;
                std::cin >> input;
                if (input == "exit")
                {
                        loop = false;
                }
        }

}

//-----実行処理-----------------------------------------------------------------------
void ServerAssignment01::Execute()
{

        WSADATA wsaData;
        // TODO 01_01 課題作成する事
        // WinsockAPIを初期化(WSAStartup関数)
        // バージョンを指定する場合MAKEWORDマクロ関数を使用する
```

課題 01_1 server 側　01ServerAssignment.cpp 完成版

```
	int wsaStartUp = WSAStartup(MAKEWORD(2, 2), &wsaData);
	if (wsaStartUp != 0)
	{
		// 初期化失敗
		std::cout << "01_01 ERR!" << std::endl;
		return;
	}

	// TODO 01_02 課題作成する事
	// サーバの受付設定
([addr.sin_family],[addr.sin_port],[addr.sin_addr.S_un.S_addr]の設定)
	struct sockaddr_in addr;
	addr.sin_family = AF_INET;
	addr.sin_port = htons(7000);
	addr.sin_addr.S_un.S_addr = INADDR_ANY;//"0.0.0.0"

	// TODO 01_03 課題作成する事
	// ソケットの作成　　(socket関数でのソケット作成：インターネット、UDP
接続設定)
	SOCKET sock = socket(AF_INET, SOCK_DGRAM, 0);  //UDP
	if (sock == INVALID_SOCKET) {
		std::cout << "01_03 ERR!" << std::endl;
		// 9.WSAの解放
		WSACleanup();
		return;
	}

	// TODO 01_04 課題作成する事
	// ノンブロッキング設定(ioctlsocket関数でのノンブロック指定)
	u_long mode = NONBLOCKING_MODE;
	int m = ioctlsocket(sock, FIONBIO, &mode); //ノンブロッキングに
	if (m != 0) {
		std::cout << "01_04 ERR!" << std::endl;
		return;
	}
```

課題 01_1 server 側　01ServerAssignment.cpp 完成版

```
	// TODO 01_05 課題作成する事
	// ソケットと受付情報を紐づける(bind関数)
	int r = bind(sock, reinterpret_cast<sockaddr*>(&addr), sizeof(addr));
	if (r != 0)
	{
		std::cout << "01_05 ERR!" << std::endl;
		return;
	}

	std::cout << "Server Initialize OK." << std::endl;

	// クライアントからの受付処理
	int size = sizeof(struct sockaddr_in);

	// サーバ側からコマンド入力で終了されるまでループする。
	// キーボードでexitを入力するとループを抜けるための別スレッドを用意
	std::thread th(&ServerAssignment01::Exit, this);

	do {
		char buffer[BUFFER_SIZE];
		int len = sizeof(sockaddr);
		int size = 0;

		// TODO 01_06 課題作成する事
		// データ受信(recvfrom関数)
		size = recvfrom(sock, buffer, sizeof(buffer), 0, //ここまで引数
はTCPと同じ)
			reinterpret_cast<sockaddr*>(&client), &len); //5番目、6
番目が増えた

		if (size > 0)//何か受信した！
		{
			std::cout << buffer << std::endl;

			// TODO 01_07 課題作成する事
			// データ送信（sendto関数で送り返し）
```

課題 01_1 server 側　01ServerAssignment.cpp 完成版

```
                        size = sendto(sock, buffer,
static_cast<int>(strlen(buffer) + 1), 0, //send同様

        reinterpret_cast<sockaddr*>(&client),static_cast<int>(sizeof(sockaddr)))
;

                        if (size <= 0)
                        {
                                std::cout << "send error:" << WSAGetLastError()
<< std::endl;
                        }
                }
                else
                {
                        if (WSAGetLastError() != WSAEWOULDBLOCK) //エラーか来な
かっただけかの判断
                        {
                                std::cout << "recvfrom failed. error code : " <<
WSAGetLastError() << std::endl;
                        }
                }
        } while (loop);
        th.join();

        // TODO 01_08 課題作成する事
        // サーバソケット切断(closesocket関数)
        r = closesocket(sock);
        if (r != 0) {
                int err = WSAGetLastError();
                std::cout << "01_08ERR! error_code:" << err << "." << std::endl;
        }

        // TODO 01_09 課題作成する事
        // WSA終了(WSACleanup関数)
        int wsaCleanup = WSACleanup();
        if (wsaCleanup != 0)
```

課題 01_1 server 側　01ServerAssignment.cpp 完成版

```
	{
		std::cout << "Cleanup WinsockAPI Failed." << std::endl;
	}else {
		std::cout << "Cleanup WinsockAPI Success." << std::endl;
	}
}
```