03ServerAssignment.cpp　完成コード

```
#include "03ServerAssignment.h"

//-----定数--------------------
#define BUFFER_SIZE            2048
#define PORT 7000
#define XYZ000       DirectX::XMFLOAT3(0.0f,0.0f,0.0f)
//-------------------------------------------------------------
// サーバ側コマンド入力スレッド
void ServerAssignment03::Exit()
{
          while (loop) {
                    std::string input;
                    std::cin >> input;
                    if (input == "exit")
                    {
                              loop = false;
                    }
          }
}

//----実行処理-----------------------------------------------------------------
void ServerAssignment03::Execute()
{
          // WinsockAPIを初期化
          WSADATA wsaData;
          if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
          {
                    // 初期化失敗
                    std::cout << "WSA Initialize Failed." << std::endl;
                    return;
          }

          // サーバの受付設定
          struct sockaddr_in addr {};
          addr.sin_family = AF_INET;
          addr.sin_port = htons(PORT);
```

03ServerAssignment.cpp　完成コード

```
addr.sin_addr.S_un.S_addr = INADDR_ANY;//"0.0.0.0"

// ソケットの作成
SOCKET sock = socket(AF_INET, SOCK_DGRAM, 0);
if (sock == INVALID_SOCKET) {
		std::cout << "Create Socket Failed." << std::endl;
		// 9.WSAの解放
		WSACleanup();
		return;
}

// ノンブロッキング
u_long mode = 1;
if (ioctlsocket(sock, FIONBIO, &mode) != 0)
{
		std::cout << "Nonblocking Mode Failed." << std::endl;
		return;
}

// ソケットと受付情報を紐づける
if (bind(sock, reinterpret_cast<sockaddr*>(&addr), sizeof(addr)) != 0)
{
		std::cout << "Bind Failed." << 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(&ServerAssignment03::Exit, this);

do {
```

03ServerAssignment.cpp　完成コード

```
		sockaddr_in temp{};
		char buffer[BUFFER_SIZE]{};
		int len = sizeof(sockaddr);
		int size = 0;

		//-----------------------------------------
		// TODO 03_01
		// 受信(recvfrom)を行い、受信データをbufferに保存
		// 送信者の情報はtempに保存
		size = recvfrom(sock, reinterpret_cast<char*>(&buffer), sizeof(buffer), 0,
			reinterpret_cast<sockaddr*>(&temp), &len);

		if (size > 0)//何か受信した
		{
			//-------------------------------------------------------------
			// TODO 03_02
			// if(1)はダミーコード
			//課題は、HasSameData関数を使用して判断するコードに修正しなさい
			// HasSameDataの仕様は関数を読み込むこと
			//　　リストと今受信した相手情報を比べ、同じ物が無ければ
			if(!HasSameData(clients,temp)/*課題03_02登録データに同じものが無ければに変更*/)
				{//送信者が登録リストにない＝新規クライアントなので以下の処理を行う
				// 新規クライアントの場合
				// 受信データからネットワークタグを取得
				NetworkTag type;
				memcpy_s(&type, sizeof(short), buffer, sizeof(short));

				// 取得したタグをもとに処理を分岐
				switch (type)
				{
					case NetworkTag::Login:
					{
						// Loginタグの場合
```

03ServerAssignment.cpp　完成コード

```
                                        // loginに受信データをコピーする
                                        PlayerLogin login{};
                                        memcpy_s(&login, sizeof(PlayerLogin),
buffer, sizeof(PlayerLogin));

                                        // IDを割り振り次の接続者のためにIDを加算し
ておく
                                        login.id = giveID;
                                        ++giveID;

                                        // プレイヤー作成
                                        Player player{};
                                        player.id = login.id;
                                        player.position = XYZ000;
                                        player.angle = XYZ000;
                                        player.state = Player::State::Idle;

                                        // 新規クライアントを追加
                                        Client* newClient = new Client;
                                        newClient->addr = temp;
                                        newClient->player = player;
                                        clients.emplace_back(newClient);
                                        std::cout << "new client insert." <<
std::endl;

                                        for (Client* client : clients)
                                        {//全クライアントだけループ
                                                //-----------------------------
---------------

                                                // TODO 03_03
                                                // 全クライアントにlogin通知
(sendto)
                                                sendto(sock,
reinterpret_cast<char*>(&login),
```

```
            static_cast<int>(sizeof(PlayerLogin)),
                                                                    0,
            reinterpret_cast<struct sockaddr*>(&client->addr),
            static_cast<int>(sizeof(sockaddr_in)));


                                                                    //------------------------------
---------
                                                                    // TODO 03_04
                                                                    // 作成したplInfoを送信者に送信
(sendto)
                                                                    //もし、for文でループしているク
ライアントリストの
                                                                    if (client-
>addr.sin_addr.S_un.S_addr !=
            temp.sin_addr.S_un.S_addr) {//アドレスが違うとき：ログインユーザー以外
                                                                        //上から移動(他のユー
ザー情報を作り)
                                                                        PlayerInformation
plInfo{};
                                                                        plInfo.cmd =
NetworkTag::Sync;
                                                                        plInfo.id = client-
>player.id;
                                                                        plInfo.position =
client->player.position;
                                                                        plInfo.angle = client-
>player.angle;
                                                                        plInfo.state = client-
>player.state;
                                                                        //新しく来たユーザーに
送る
```

03ServerAssignment.cpp　完成コード

```
                                                                sendto(sock,
reinterpret_cast<char*>(&plInfo),
        sizeof(PlayerInformation),
                                                                        0,
        reinterpret_cast<sockaddr*>(&temp),
        sizeof(sockaddr_in)
                                                                );
                                                        }
                                                }

                                                break;
                                        }
                                        default:
                                        {
                                                break;
                                        }
                                }
                        }
                        else
                        {
                                // 既存のクライアントの場合
                                // 受信データからネットワークタグを取得
                                NetworkTag type;
                                memcpy_s(&type, sizeof(short), buffer, sizeof(short));

                                switch (type)
                                {
                                        case NetworkTag::Sync:
                                        {
                                                // Syncタグの場合
                                                // 受信データをplInfoにコピーする
                                                PlayerInformation plInfo{};
                                                memcpy_s(&plInfo,
```

03ServerAssignment.cpp　完成コード

```
sizeof(PlayerInformation), buffer, sizeof(PlayerInformation));

                                                        // 受信データを表示
                                                        std::cout << "position :(" <<
plInfo.position.x << "," << plInfo.position.y << "," << plInfo.position.z << ")" << std::endl;
                                                        std::cout << "angle :(" << plInfo.angle.x
<< "," << plInfo.angle.y << "," << plInfo.angle.z << ")" << std::endl;
                                                        std::cout << "state : " <<
static_cast<int>(plInfo.state) << std::endl;

                                                        // 既存クライアント全員に送信
                                                        for (Client* client : clients)//クライアン
ト数だけループ
                                                        {
                                                                // サーバのプレイヤー情報更新
                                                                if (client->player.id ==
plInfo.id)
                                                                {//送信者だったら　サーバーのプ
レイヤー情報を情報更新して
                                                                        client-
>player.position = plInfo.position;
                                                                        client->player.angle =
plInfo.angle;
                                                                        client->player.state =
plInfo.state;
                                                                }
                                                                else
                                                                {//送信者以外には配信する
                                                                        //----------------------
----------------------
                                                                        // TODO 03_05
                                                                        // プレイヤー情報を送
信する(sendtoでplInfoを送る)
                                                                        size = sendto(sock,
reinterpret_cast<char*>(&plInfo),
```

03ServerAssignment.cpp　完成コード

```
            sizeof(PlayerInformation),
                                                                    0,

            reinterpret_cast<sockaddr*>(&client->addr),

            static_cast<int>(sizeof(sockaddr_in)));
                                                                    //ここまで03_05
                                                                    if (size < 0)
                                                                    {
                                                                            std::cout
<< "sendto failed. error code : " << WSAGetLastError() << std::endl;
                                                                    }
                                                            }
                                                    }
                                                    break;
                                            }
                                            case NetworkTag::Logout:
                                            {
                                                    // ログアウトデータの場合
                                                    // 受信データをlogoputにコピーする
                                                    PlayerLogout logout{};
                                                    memcpy_s(&logout, sizeof(PlayerLogout),
buffer, sizeof(PlayerLogout));

                                                    for (auto it = clients.begin(); it !=
clients.end();)
                                                    {
                                                            // 送信者とidが同じとき
                                                            if ((*it)->player.id ==
logout.id) {
                                                                    // データを削除
                                                                    it =
clients.erase(it);
                                                            }
                                                            else {
                                                                    //----------
```

03ServerAssignment.cpp　完成コード

```
-----------------------
                                                                        // TODO 03_06
                                                                        // logoutデータを送信
者以外に送信(sendtoでlogoutを送る)
                                                                        size = sendto(sock,
reinterpret_cast<char*>(&logout),
            sizeof(PlayerInformation),
                                                                                    0,
            reinterpret_cast<sockaddr*>(&(*it)->addr),
            static_cast<int>(sizeof(sockaddr_in)));
                                                                        ++it; // 次の要素に進
む
                                                                    }
                                                                }
                                                                break;
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                            else
                                            {
                                                if (WSAGetLastError() != WSAEWOULDBLOCK)
                                                {
                                                    std::cout << "recvfrom failed. error code : " <<
WSAGetLastError() << std::endl;
                                                }
                                            }
            } while (loop);
            th.join();

            // クライアント終了処理
            for (Client* client : clients)
```

```
	{
		if (client != nullptr)
		{
			delete client;
			client = nullptr;
		}
	}

	// サーバソケットの切断
	if (closesocket(sock) != 0) {
		int err = WSAGetLastError();
		std::cout << "Close Socket Failed.error_code:" << err << "." << std::endl;
	}

	// WSA終了
	if (WSACleanup() != 0)
	{
		std::cout << "Cleanup WinsockAPI Failed." << std::endl;

	}
	std::cout << "Cleanup WinsockAPI Success." << std::endl;
}

//----アドレス確認-----------------------------------------------------------------
//所持クライアント情報と、新規クライアントを比べて既に存在するかをチェック
bool ServerAssignment03::HasSameData(const std::vector<Client*>& vec, const sockaddr_in& target)
{
	for (const Client* client : vec)
	{
		if (client->addr.sin_addr.S_un.S_addr == target.sin_addr.S_un.S_addr &&
			client->addr.sin_port == target.sin_port)
		{
			return true;
		}
	}
	return false;
```

```
}
```