모든 데이터(패킷, 문자열, 시그널)를 보내기전에는 항상 header를 먼저 보냅시다!

// 헤더 타입

typedef struct header {

int type;

int len;

}Header;

type: 0(문자열), 1(구조체), 2(시그널)

// 패킷 타입

typedef struct player\_packet {

char info;

short position;

char cards[4];

char dice;

short infer;

char clue;

}Player\_packet;

※ 기본적으로 ‘단계’ 정보는 2가지 경우로 나눈다. 각 경우는 0, 1로 표현한다.

0 : 주사위 굴림, 위치 이동

1 : 추리, 단서 제출

※ 사용하는 시그널

2 : SIG\_TURN

3 : SIG\_WAIT

4 : SIG\_INFR

5 : SIG\_DONE

**/\* 공용함수 \*/**

int packet\_send(int sock, char \*packet, int \*type)

// 패킷 전송 함수

// 성공 : 0

// 실패 : -1

***// packet\_send 함수 진행 과정***

// write

int packet\_recv(int sock, char \*packet, int \*type)

// 패킷 수신 함수

// 패킷이 들어올 때까지 대기함

// 성공 : 수신된 패킷

// 실패 : -1

***// packet\_recv 함수 진행 과정***

// read =>

**/\* server function \*/**

1. int server\_open()

// 서버 생성 함수

// 성공 : 서버 소켓 파일 디스크립터

// 실패 : -1

2. int\* server\_accept()

// 서버에서 클라이언트를 받는 함수

// 성공 : 클라이언트 4명의 파일 디스크립터가 들어있는 int형 배열의 시작 주소

// 실패 : NULL

3. char\* game\_set\_answer()

// 3장의 정답 카드 배열을 동적할당해서 리턴하는 함수

// 성공 : 정답 카드 배열의 시작주소

// 실패 : NULL

4. Player\_packet\*\* game\_init(int \*players)

// 게임 초기화 함수

// 서버에서 관리하는 플레이어 4명의 패킷(Player\_packet타입의 구조체)을 동적할당 한 뒤,

// 4명의 패킷을 초기화 하고 각 클라이언트에게 전송

// 4명 모두에게 패킷을 잘 받았다는 응답을 받으면 동적할당한 패킷배열의 시작주소를 리턴

// 성공 : 서버가 관리할 패킷배열의 시작주소

// 실패 : NULL

// ***game\_init 함수 진행 과정(아래 함수들의 호출로 과정이 진행됨)*** :

// game\_init\_cards ->

// game\_init\_players ->

// game\_route\_packets

int game\_init\_cards(Player\_packet \*\*player\_packets, char \*answer)

// 각 클라이언트 패킷에 16장의 카드를 무작위로 4장씩 분배하여 세팅하는 함수

// 성공 : 0

// 실패 : -1

int game\_init\_players(Player\_packet \*\*player\_packets)

// 각 클라이언트 패킷에 플레이어의 맵상의 위치정보를 (1,1)로 세팅하고,

// 플레이어 식별 비트를 세팅하는 함수

// 성공 : 0

// 실패 : -1

int game\_route\_packets(Player\_packet \*\*player\_packets, int \*players)

// 모든 클라이언트에게 패킷을 전송하는 함수

// 성공 : 0

// 실패 : -1

***// game\_route 함수 진행 과정 :***

// packet\_send 4번 반복

int game\_route\_string(char \*string, int \*player)

// 모든 클라이언트에게 문자열을 전송하는 함수

// 성공 : 0

// 실패 : -1

// game\_init 함수종료!!

/\* 서버 코드에서 루프가 시작됩니다… \*/

/\* 아래의 함수들의 호출이 반복적으로 실행됩니다… \*/

/\* 해당 루프가 반복되며 게임이 진행됩니다… \*/

/\* 루프 시작 \*/

// packet\_send(해당 턴인 클라이언트에게는 SIG\_TURN, 나머지에게는 SIG\_WAIT 전달) ->

// game\_roll\_and\_go =>

// game\_infer =>

// gama\_next\_turn =>

/\* 루프의 끝 \*/

5. int game\_roll\_and\_go(Player\_packet \*\*player\_packets, int \*players)

// 턴인 클라이언트로부터 주사위값, 선택값, 위치값이 세팅된 패킷을 받은 뒤, 서버에서 관리하던 // 4개의 플레이어 패킷에 해당 값들을 세팅해서 모든 클라이언트에게 전송

// ***game\_roll\_and\_go 함수 진행 과정:***

// packet\_recv ->

// game\_set\_dice ->

// game\_set\_player ->

// game\_set\_phase ->

// game\_route

※ 아래 함수에서는 미리 하나의 패킷만큼의 메모리를 할당받은 뒤 거기에 턴클의

패킷을 저장했다고 가정

int game\_set\_dice(Player\_packet \*\*player\_packets, Player\_packet\* player\_packet)

// 모든 클라이언트 패킷에 주사위값, 선택값을 세팅하는 함수

// player\_packet인자는 packet\_recv함수로부터 받은 턴클의 패킷

// 해당 패킷을 파싱해서 주사위값, 선택값을 추출한 뒤 서버가 가지고 있는

// 모든 클라이언트의 패킷에 세팅해줌

// 성공 : 0

// 실패 : -1

int game\_set\_position(Player\_packet \*\*player\_packets, Player\_packet\* player\_packet)

// 모든 클라이언트 패킷에 플레이어의 좌표값을 세팅하는 함수

// game\_set\_dice함수와 진행과정은 같음

// 성공 : 0

// 실패 : -1’

int game\_set\_phase(Player\_packet \*\*player\_packets)

// 모든 클라이언트 패킷에 알맞은 단계를 세팅하는 함수

// 4개의 클라이언트 패킷 중, 턴 클라이언트의 패킷의 ‘단계’ 부분은 1(추리,단서과정)로 // 세팅하고, 나머지 클라이언트 패킷의 ‘단계’ 부분은 0으로 세팅하는 함수

// 따라서 턴 클라이언트 식별자(turn\_player)가 반드시 인자로 넘어와야함

// turn\_player 변수는 루프 전에 선언되어 있음

// 성공 : 0

// 실패 : -1

// game\_route 진행

// game\_infer함수로 넘어감

6. game\_infer(infer함수가 끝나면 패킷 초기화 진행)

// 우선 턴 클라이언트에게 추리 신호를 보내서 추리정보가 담긴 패킷을 받는다

// 서버가 관리하는 4개의 클라이언트 패킷 중에서 턴 클라이언트의 패킷을 제외한 나머지 3개의 // 패킷에 해당 추리정보를 세팅한 뒤, 각 클라이언트에게 세팅된 패킷을 순서대로 보내고 단서 // 응답을 받아서 처리한다

/\* 추리 과정 \*/

// 2. 턴플에게 SIG\_INFR 전송

// 3. 턴플이 추리 정보가 담긴 패킷 전송

// 모든 플레이어의 패킷에 추리 정보를 반영

// 4. int value = SIG\_TURN 설정 후 for문으로 n-1번 돌며 value(시그널)전송

// 5. 서버 입장에서 단서가 0인 패킷을 수신하면 다음 플레이어에게 SIG\_TURN 전송

// 5-1. value가 SIG\_TURN : 턴클에게 SIG\_TURN 을 전송하고 단서 wait

// 5-1-1. 턴클에게 단서가 없다면 : 서버 입장에서 단서가 0인 패킷을 수신하면 ‘아무도 단서가 없다’고 출력 후 턴 종료

// 5-1-2. 턴클에게 단서가 있다면 : 서버 입장에서 단서가 0이 아닌 패킷을 수신하면 route로 모두에게 단서 전송

// 5-2. Value가 SIG\_WAIT : 턴클에게 SIG\_WAIT을 전송하고 나머지 클에게 단서 패킷을 전송

// ***game\_infer 함수 진행 과정*** :

// 아직 못함

int game\_set\_infer(Player\_packet \*\*player\_packets)

// 각 클라이언트 패킷에 추리정보를 세팅하는 함수

// 성공 : 0

// 오류 : -1

int game\_set\_clue(Player\_packet \*\*player\_packets)

// 각 클라이언트 패킷에 단서값을 세팅하는 함수

// 성공 : 0

// 오류 : -1

int game\_ROOM\_OF\_TRUTH(Player\_packet \*\*player\_packets, char \*answer, int player)

// 클라이언트의 추리와 서버에 저장되어있던 정답을 비교하여 정답이면 게임 종료

// 틀릴경우 해당 클라이언트를 게임에서 제외(game\_infer에서 호출)

***// game\_ROOM\_OF\_TRUTH 함수 진행 과정*** :

// 턴클의 추리와 서버의 정답을 비교 =>

// 정답인 경우 : return\_zero

// 오답인 경우 : game\_player\_out

int end\_game(Player\_packet \*\*player\_packets, int \*players,

char \*answer, int server\_sock)

// ***end\_game 함수 진행 과정*** :

// 문자열을 라우팅(게임이 끝났습니다) =>

// 패킷 할당 해제 =>

// 플레이어 할당 해제 =>

// 정답 카드 해제 =>

// 파일 디스크립터 close

// 서버 소켓 해제

int game\_player\_out(int player)

// 클라이언트가 게임에서 제외되는 상황을 처리

// packet의 주소에 해당하는 메모리를 해제한 뒤,

// 포인터의 위치를 앞으로 재조정

7. int game\_next\_turn(Player\_packet \*\*player\_packets)

// 성공 : 다음 턴의 인덱스(플레이어 식별 비트)

// 실패 : -1

// ***game\_next\_turn 함수 진행 과정*** :

// game\_init\_player\_data =>

// game\_set\_turn

int game\_init\_player\_data(Player\_packet \*\*player\_packets)

// 패킷의 주사위값, 선택값, 추리, 단서, 단계 부분 초기화

// 성공 : 0

// 실패 : -1

int game\_set\_turn(Player\_packet \*\*player\_packets)

// 모든 클라이언트의 턴 비트를 변경, PLAYER\_ISTURN 매크로 활용함

// 성공 : 다음 턴의 인덱스(플레이어 식별 비트)

// 실패 : -1

**/\* client function \*/**

1. int client\_connect()

// 소켓 생성 후 접속

// 성공 : 서버와 연결하는 소켓

// 실패 : -1

2. int game\_update(int sock)

// 서버로부터 패킷을 받아 화면을 업데이트

// 성공 : 0

// 실패 : -1

***// game\_update 함수 진행 과정:***

// packet\_recv =>

// ui\_update

int ui\_update(Player\_packet \*player\_packet)

// 맵의 말 위치, 자신의 카드 4장, 히스토리, 진행사항 화면에 업데이트

// 성공 : 0

// 실패 : -1

***// ui\_update 함수 진행 과정:***

int change\_player\_position(WINDOW \*window, Player\_packet \*player\_packet)

// 맵의 말 하나의 위치 조정 함수

// 성공 : 0

// 실패 : -1

int show\_my\_cards(WINDOW \*window, Player\_packet \*player\_packet)

// 자신의 카드 4장을 화면에 출력

// 성공 : 0

// 실패 : -1

int show\_history(WINDOW \*window, Player\_packet \*player\_packet)

// 화면의 히스토리 부분 출력

// 성공 : 0

// 실패 : -1

int show\_game\_log(WINDOW \*window, Player\_packet \*player\_packet)

// 화면의 게임진행 부분 출력

// 성공 : 0

// 실패 : -1

/\* 이후 각 클라이언트가 packet\_recv로 대기하고 있으면, 서버는 SIG\_TURN 혹은 SIG\_WAIT 신호를 보낼 것이다. \*/

/\* SIG\_WAIT이 온 경우 \*/

해당 턴인 플레이어를 대기함

/\* SIG\_TURN이 온 경우 \*/

자신의 턴을 시작한다.

/\* 턴 시작 \*/

3. int roll\_and\_go(int sock)

// 주사위값, 선택값, 위치값을 패킷에 설정한 뒤 서버에 전송

// 성공 : 0

// 실패 : -1

// ***roll\_and\_go 함수 진행 과정:***

// roll\_dice =>

// return\_player\_choice =>

// return\_player\_position =>

// set\_dice(주사위값, 선택값 세팅) =>

// set\_player(위치값 세팅) =>

// packet\_send

int roll\_dice()

// 주사위값을 생성하는 함수

// 성공 : 주사위값

// 실패 : -1

int return\_player\_choice(int dice\_value)

// 선택값을 리턴하는 함수

// 성공 : 선택값

// 실패 : -1

int return\_player\_position(int choice\_value, int \*y, int \*x)

// 위치값을 리턴하는 함수

// 성공 : 0

// 실패 : -1

int set\_dice\_in\_packet(Player\_packet \*player\_packet)

// 주사위값, 선택값을 패킷에 세팅

// 성공 : 0

// 실패 : -1

int set\_player(Player\_packet \*player\_packet)

// 위치값을 패킷에 세팅

// 성공 : 0

// 실패 : -1

4. int game\_infer()

// 추리정보를 서버로 보내고 단서 송수신 과정을 처리하는 함수

// 서버는 턴클에게는 SIG\_INFR를, 나머지 클에게는 SIG\_WAIT을 보냄

// SIG\_INFR(턴클라이언트) :

// 신호를 받으면 추리정보가 담긴 패킷을 서버에 보냄

// 그리고 다시 SIG\_WAIT 또는 SIG\_TURN 신호를 기다림

// SIG\_WAIT(나머지 클라이언트) :

// SIG\_TURN 혹은 SIG\_DONE 신호를 기다림kj

// 성공 : 0

// 실패 : -1

// ***game\_infer 함수 진행 과정*** :

// packet\_recv => SIG\_INFR 혹은 SIG\_WAIT 두가지로 분기됨

/\* SIG\_INFR가 올 경우의 진행 과정(턴클라이언트) \*/

// set\_infer =>

// packet\_send =>

// packet\_recv =>

// SIG\_WAIT : 다른 플레이어가 단서를 냈으니 그거 받아라(=> packet\_recv)

// SIG\_TURN : 아무도 단서를 안냈다. 너의 카드 중 한장을 보내라(=> packet\_send)

/\* SIG\_WAIT이 올 경우의 진행 과정(나머지 클라이언트) \*/

// packet\_recv =>

// SIG\_TURN: 너가 단서 보낼 차례니 단서 보내라(=> packet\_send)

// SIG\_DONE: 다른 플레이어가 단서를 냈으니 너는 그냥 넘어가라.