B5 – Advanced C++ - R-type

B-CPP-500

R-type

Let the people speak!

R-type

Binary Communication Protocol

Login

To connect to the server, you just need to connect to the ip address and port of the server. The server will then send you a reply containing the player's uuid for this game.

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

- The rfc code allows customers to know what type of data they are receiving. Each request has its own rfc code to be able to differentiate them. In this case the rfc code will be equal to SC_PLAYER_UUID.

- The size contained in the header defines the size that must be read from the socket to be able to retrieve the end of the packet.

```
typedef struct player_uuid {
        header_t header;
        char uuid[Rtype::Size::uuidLength];
}
```

- At each connection to the server, the client will receive its uuid which is specific to his gaming session. It must be used for each request made on the udp server during the game.

Get Room

To be able to retrieve the list of all the rooms currently available. I.e., the rooms that can still accept new connections. You must send a packet to the server, which contains just a header with the code CS_GET_ROOM, and the size must be set to zero.

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

You will first receive a header containing SC_OK if the request was successful, or SC_ERROR otherwise. In case the request is launched correctly, you will receive each room in the form of the following structure:

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

- The rfc code allows customers to know what type of data they are receiving. Each request has its own rfc code to be able to differentiate them. In this case the rfc code will be equal to SC_ROOM.

- The size contained in the header defines the size that must be read from the socket to be able to retrieve the end of the packet.

```
typedef struct get_room_s {
    header_t header;
    char uuid[Rtype::Size::uuidLength];
    char name[Rtype::Size::nameLength];
    unsigned short port;
    short place;
    short numberOfPlayer;
    short bomb;
    short life;
}get_room_t;
```

- The uuid is specific to the room, it allows to identify the room in question and to be able to interact with it.
- The name is the name of the room, be careful, two rooms can have the same name.
- The port is the one to which the client must connect in order to interact with the udp server that contains the room.
- The Place defines the number of players that the room can contain at most.
- The numberOfPlayer sets the number of players currently in the room.
- The bomb sets the maximum number of bombs the user can have during the game.
- The bomb sets the maximum number of lifes the user can have during the game.

Create Room

In order to be able to create rooms, you have to send a structure, which contains all the necessary information for its creation. The structure must be defined as follows:

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

- The rfc code allows the server to understand which command the client wants to run. It must be set to CS_CREATE_ROOM.
- The size contained in the header defines the size that must be read from the socket to be able to retrieve the end of the packet.

```
typedef struct create_room_s {
    header_t header;
    char name[Rtype::Size::nameLength];
    short place;
    short bomb;
    short life;
}create_room_t;
```

- The name the user wants to give to the room.
- The maximum number of places that the room can accommodate.
- The maximum number of bombs that players will be able to get.
- The number of lives the players will have during the game.

You will first receive a header containing SC_OK if the request was successful, or SC_ERROR otherwise

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

- The rfc code allows customers to know what type of data they are receiving. Each request has its own rfc code to be able to differentiate them. In this case the rfc code will be equal to SC_OK or SC_ERROR.

- The size contained in the header defines the size that must be read from the socket to be able to retrieve the end of the packet

If the request worked correctly you will receive a second package with the rfc code ROOM_LOCATION. This packet allows you to know the port on which the room is hosted but also to know its ID.

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;

typedef struct room_location_s {
    header_t header;
    char uuid[Rtype::Size::uuidLength];
    unsigned short port;
}room_location_t;
```

Join Room

To be able to join a room, you must first have the uuid of the room in question, which you can get by making a "get_room" request. Once you get the uuid of the room, you just have to send a join_room request to the server.

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

- The rfc code allows the server to understand which command the client wants to run. It must be set to CS_JOIN_ROOM.

- The size contained in the header defines the size that must be read from the socket to be able to retrieve the end of the packet.

```
typedef struct join_room_s {
        header_t header;
        char uuid[Rtype::Size::uuidLength];
} join_room_t;
```

- The uuid is specific to the room, it allows to identify the room in question and to be able to interact with it.

You will first receive a header containing SC_OK if the request was successful, or SC_ERROR otherwise

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

- The rfc code allows customers to know what type of data they are receiving. Each request has its own rfc code to be able to differentiate them. In this case the rfc code will be equal to SC_OK or SC_ERROR.

- The size contained in the header defines the size that must be read from the socket to be able to retrieve the end of the packet

If the request worked correctly you will receive a second package with the rfc code ROOM_LOCATION. This packet allows you to know the port on which the room is hosted but also to know its ID.

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

```
typedef struct room_location_s {
    header_t header;
    char uuid[Rtype::Size::uuidLength];
    unsigned short port;
}room_location_t;
```


Leave Room

To leave a room, simply send a "leave room" request to the server. The uuid of the room must also be specified in the request.

```
typedef struct header_s {
    Rtype::Rfc::TcpCode type;
    short size;
}header_t;
```

- The rfc code allows the server to understand which command the client wants to run. It must be set to CS_LEAVE_ROOM.
- The size contained in the header defines the size that must be read from the socket to be able to retrieve the end of the packet.

```
typedef struct join_room_s {
        header_t header;
        char uuid[Rtype::Size::uuidLength];
} join_room_t;
```

If the request was successful you will receive a header with the code SC_OK, otherwise the code will be SC_ERROR.