Rock, Paper, Scissors

We would like two programs (a referee and a player ­ there will be two players running at the same time) that interact over the Internet to play the "Rock, Paper, Scissors" game. In this game, two players secretly choose one item out of the set of rock, paper, and scissors. They then reveal their choice. A referee decides who wins as follows:

* Paper beats rock (by covering it).
* Rock beats scissors (by smashing it).
* Scissors beats paper (by cutting it).
* Matching choices are a draw.

The winning player gets a point. In a draw, no points are awarded.

The referee program (the "server") should create and bind a network socket with a random port chosen. Must take port and rounnds as the command­line arguments. The port number should be displayed. Then the referee should wait for two players to connect and then send data to the players indicating that the game may begin.

The player program (the "client") must take a server machine name and a port number as command­line arguments. It should create a network socket and connect to the referee process . (If there is no referee process to connect to, the program should exit with an error message.) A player then should prompt the local user to enter a choice. The choice is sent to the referee. After the referee receives two choices, it should decide which player has won the round and send information about the round back to both players so that they may display the results to the local user.

The system must implement the following network protocol. The player programs must send the string "READY" to the referee when they are ready to make a choice. The referee should then respond to each player with the string "GO". Thereafter, the players alternate sending an item choice and receiving the results of the round. (The format of this information is your choice.) When a user wishes to exit, the player program should send the "STOP" string to the referee. The referee should respond with the string "STOP" to both players followed by summary game information (number of rounds won by each player, etc., again the format of this information is your choice). The players should display this information to the local user and then exit. The referee should return to waiting for the next pair of player connections. (I.e., the referee process is intended to run indefinitely.)

Write programs named **referee** and **player** that implement the referee and player programs.

Here is an example of possible referee and player sessions running on (fictitious) machines named

newton, galileo, and einstein. (User input is shown in bold.

On newton On galileo

$ **./referee** $ **./player newton 2345**

Referee is using port You are player 1.

2345

Referee is waiting for 0: Exit players 1: Rock

Player 1 has connected 2: Paper

Player 2 has connected 3: Scissors

P1 choice: Rock Enter Choice: **1** P2 choice: Scissors Round 1:

P1 wins Player 1: Rock

P1 choice: Rock Player 2: Scissors P2 choice: Exit You win!

Game has ended

Referee is waiting for 0: Exit players 1: Rock

2: Paper

3: Scissors

Enter Choice:**1**

Game has ended

Final Score

Player 1: 1

Player 2: 0

$

On einstein

$ **./player newton 2345** You are player 2.

0: Exit

1: Rock

2: Paper

3: Scissors

Enter Choice: **3**

Round 1:

Player 1: Rock layer 2: Scissors Sorry, you lose

0: Exit

1: Rock

2: Paper

3: Scissors

Enter Choice: **0**

Game has ended

Final Score

Player 1: 1

Player 2: 0

$

Note that you should test your programs using different machines for each process. It should allow the user to give the machine name and port number of a referee process and connect to it using a connect button.

* It should allow the user to indicate their item choice via mouse click (i.e., the user should not need to type the item choice), then send it to the referee using a send button.
* It should display the referee result.
* It should have a quit button that causes the player program to display the final results and require another button click to actually quit the program.