2006/2007 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
UniClue

(This problem uses an interactive server.)

Clue is a Hasbro board game where three to six players try to solve the circumstances (killer, weapon, and location) of a murder. In preparation for the annual International Collegiate Clue Contest, Swamp County College will produce a computer program called “UniClue” for training individual players in the techniques necessary for solving the crime. UniClue will implement a computer-based strategy for play against humans, offering a modest level of competition for humans-in-training. Your team is to implement the computer-based opponent as a client in a client-server relationship. An all-knowing server will respond to your program’s inputs, directing the play of the game.

In Clue, there are six suspects, six weapons, and nine rooms. Each suspect, weapon, and room is represented on a single card. Prior to play, an unknown suspect card, unknown weapon card, and unknown room card are drawn, then placed in an envelope. All remaining cards are shuffled together and dealt randomly to the players. In the real game of Clue, players roll a die and move tokens around a board, then query suspected combinations of circumstances (suspect, weapon, and room). A player can only query the room that he or she occupies. The remaining players respond in clockwise order to each suspicion, privately offering up to the querying player a single card (an alibi) matching one of the circumstances, or replying publicly "No alibi" when none of the cards in the player’s hand matches any one of the suspect, weapon, or room being queried. As soon as an alibi is produced, rolling, movement, and query procedes to the next player. Reponses to queries slowly reveal the contents of the opponents’ hands to the player making the query (and to any other opponent observing carefully). When a player believes that he or she has enough information, he or she offers up an accusation that proposes the suspect, weapon, and room cards in the envelope. A correct accusation earns victory for the player; an incorrect accusation disqualifies the accusing player from making any further queries or accusations.

The roll of the die and movement of tokens around a game board merely introduce a level of randomness and present opportunities for defensive play. UniClue dispenses with the die roll, instead having the server supply pseudo-random room locations to your client. Furthermore, to allow a novice human opponent to improve and eventually beat UniClue, the server will supply responses to your program’s queries only, denying your program any chance to collect information available from other players’ turns. Despite this built-in disadvantage, your program is to play competently: your program must be able to form a correct accusation in fewer than 43 queries. The judges’ test cases will not present any room sequences that prevent a competent program from making a successful accusation within the query limit.

Because your program is being evaluated only for its query strategy, you will not implement any code to issue reponses to other players’ queries (the server responds as a proxy on behalf of all players). Regarding the server’s responses to queries: though random, cards are dealt in a resulting sequential order to the players. When offering up alibis, the server responds with

1. any alibi card that it has already shown your client, or if none
2. the first alibi card found through a sequential search of the cards in each player’s hand.

Program Interaction

Your program must converse with a server, issuing commands to standard output and receiving responses through standard input. There are three commands, ‘C’ to obtain the game configuration, ‘Q’ to query a suspicion, and ‘A’ to form an accusation. The ‘C’ command is a single character followed by end-of-line. The server reponds with with two lines of configuration data and a line with the room number that your client occupies. The first line is a single integer P denoting the number of players in the game. The second line lists the cards in your hand, of the form *cn*, where *c* is the card type (S=suspect, W=weapon, and R=room); *n* is the card number: 1–6 for suspects, 1–6 for weapons, and 1–9 for rooms. The number of cards in your hand depends upon the number of players in the game, with lower-numbered players possibly having more cards than higher-numbered players. Your program is always player one (1).

Problem 4
UniClue (continued)

‘Q’ and ‘A’ are of the form ‘Q*swr*’ and ‘A*swr*’ followed by end-of-line; *s* is the suspect number 1–6, *w* is the weapon number 1–6, and *r* is the room number 1–9. For the ‘Q’ command, the server supplies each other player’s response in the form *prr*, where *p* is the player number (2–P) and *rr* is either the card shown or ‘––’ indicating no alibi. Once an alibi is presented, the server responds with a new room number. If all P players have no alibi, the server reports a line for each player, then issues a new room number. For the ‘A’ command, the server replies “Victory” or “Disqualified”, then terminates.

Your program must initiate the conversation by issuing a ‘C’ command to get the configuration. No other players will make accusations during your client’s execution. Your program can issue queries only for the room that it currently occupies, but may make an accusation for any room at any time.

It is very important that your program flush standard output after issuing a command to the server. e.g.

C	`fputs("C\n",stdout); fflush(stdout);`
C++	`cout << "C\n" << flush;`
Java	`System.out.println("C"); System.out.flush();`

A sample transcript follows. Client commands are offset for clarity. The comments to the right of the transcript must not appear in your client commands and will not occur in the server’s responses; they are for description only.

Sample Transcript of Game Play

```
Server            Client   Comments

                  C        (Query configuration)
4                          (the number of players)
R2 S1 R1 W2 W1             (cards dealt to your client: room 2, suspect 1, room 1, weapon 2, weapon 1)
3                          (your client is in room 3)
                  Q333     (query S W L: suspect 3, weapon 3, room 3)
2W3                        (player 2 shows weapon 3 card to you; other players know only that a card is shown)
5                          (your client is in room 5)
                  Q555     (query S W L: suspect 5, weapon 5, room 5)
2--                        (no alibi)
3S5                        (player 3 shows suspect 5 card to you)
3                          (your client is in room 3)
                  Q353     (query S W L: suspect 3, weapon 5, room 3)
2S3
4
                  Q264     (query S W L: suspect 2, weapon 6, room 4)
2R4
7
                  Q667
2--
3S6
2
                  Q242
2W4
8
```

(Transcript continued on next page)

Problem 4
UniClue (still continued)

```
                Q268
2--
3--
4R8
6
                Q166
2--
3R6
4
                Q464
2R4                    (player 2 shows room 4 card to you again, no new information possible)
9
                Q259
2--
3--
4R9
3
                Q263
2--
3--
4W6
1
                Q411
2S4
5
                Q115
2--
3R5
7
                Q127
2--
3--
4R7
2
                A253   (accuse suspect 2 using weapon 5 in room 3)
Victory
```

(Problem description continued on next page)

Problem 4
UniClue (still continued)

Hints for Testing

You may construct a series of hands for a game manually. Execute your client program with you, the programmer, acting as the server. Let standard input come from your command line environment. Supply correct responses to your program, keeping track of card order and cards that have been shown previously.

The server process used by the judges is available to the contestants. You may use the server in two ways. In the absence of a working client program, you, the programmer, can act as the client, typing input to the server. Use the following command:

`test4` *configurationFile transcriptFile*

To test your program and its conversation with the server, use the command:

`test4` *configurationFile transcriptFile sourceFile*

where

configurationFile is described below

transcriptFile is a file that captures the server's inputs and outputs for later analysis. The server's inputs are your client's commands. Client commands are offset by two tab characters to distinguish the client vs. server dialog.

sourceFile is the source code to your client. When specified, the test4 script calls the compile script.

You can "deal a game" to this server by supplying a configuration file in the following format: The first line contains P, the number of players. The following P lines contain the players' hands, with single spaces separating the cards in each hand. The remaining lines contain single integers denoting the sequence of room numbers. Once the server runs out of specified rooms, it repeats 1–9 sequentially; omitting room numbers from the specification produces the repeating sequence 1–9, 1–9, . . . The configuration for the above transcript is in file sample4.in and is repeated here. The comments to the right of the configuration file entries must not appear in the actual file; they are here for information only.

`4`	(four players)
`R2 S1 R1 W2 W1`	(player 1's hand) (your client)
`W4 W3 S3 R4 S4`	(player 2's hand)
`R6 S5 S6 R5`	(player 3's hand)
`R8 W6 R7 R9`	(player 4's hand) (by process of elimination, the S2,W5,R3 are in the envelope)
`3`	(sequence of room numbers. . .)
`5`	
`3`	
`4`	
`7`	
`2`	
`8`	
`6`	
`4`	
`9`	
`3`	
`1`	
`5`	
`7`	
`2`	(. . .once the server runs out of rooms, it repeats 1–9 sequentially)