

PKMATCH

In this two-players card game, each card shows a picture of a monster. There are 6 monsters in the game and they are represented with the integers $\{0, 1, 2, 3, 4, 5\}$.

At the beginning of each game, each player holds 6 cards without duplicates. In other words, each player starts with all 6 monsters on hand. The game is played in rounds. In each round, each player picks a card from his/her remaining cards and shows it to the other player. If both players chose the same card, both cards are removed from the players. Otherwise, the monsters fight and one of them gets knocked-out according to a strength table described below. The knocked-out monster is removed. The game ends when either player has all his cards removed. The player who has cards when the game ends wins the game. If both players have no cards left when the game ends, both players "win", i.e. a "draw" is considered to be a "win".

The strength table indicates which monster get knocked-out when two monsters fight. Suppose monsters i and j fight, we have $S(i, j) = i$ and $S(j, i) = i$ iff monster i gets knocked-out. The following is an example of strength table. Note that $S(i, i) = i$ for all i since both monsters get knocked-out when they are the same. Furthermore, the strength table is symmetric in the sense that $S(i, j) = S(j, i)$ for all i's and j's.

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	1	2	3	4	5
2	2	2	2	3	4	5
3	3	3	3	3	4	5
4	4	4	4	4	4	5
5	5	5	5	5	5	5

Your task is to write a program to play a series of 5,000 games with Randy. A different strength table will be used in each game. Randy always uses a simple strategy: among the remaining cards, it simply randomly picks a card to fight, where each card is equally likely to be chosen. Your goal is to win as many games as possible.

Interactions

In your home directory, you can find two programs: `RANDY` and `interact.sh`. The program `RANDY` is provided by Randy. We now describe the input & output of `RANDY` and how to interact with `RANDY` using `interact.sh`.

Input/Output of `RANDY`

You can "manually" play two games with Randy by issuing the command:

```
./RANDY 2
```

`RANDY` starts the game by displaying the strength table to be used in that game. It first displays 6 lines, each line contains 6 integers. The j-th integer from the left in the i-line is $S(i-1, j-1)$



as shown in the example below. Next, `RANDY` waits for you to enter your choice of monster in the first round, which is an integer. After received your choice, `RANDY` displays its choice in the first round, waits for your choice of the second round, and so on. After the game has ended, `RANDY` displays "`-1`" if you win that game, or "`-2`" if `RANDY` wins that game, and immediate goes to the next game. After all the games have ended, `RANDY` displays the total number of games you have won.

Example

Here is the example of messages exchanged when the command

```
./RANDY 2
```

is issued. For clarity, the input to `RANDY` (in other words, messages sent by you) are shown in boldface.

```
0 1 2 3 4 5
1 1 2 3 4 5
2 2 2 3 4 5
3 3 3 3 4 5
4 4 4 4 4 5
5 5 5 5 5 5
0
4
0
5
0
2
0
3
0
1
0
0
-1
0 0 0 3 0 0
0 1 2 1 4 5
0 2 2 2 2 5
3 1 2 3 3 3
0 4 2 3 4 4
0 5 5 3 4 5
0
5
1
```



```
0
1
4
1
3
2
2
3
5
4
5
5
5
-2
1
```

Interactions between `RANDY` **and your program**

The previous example show the input and output of `RNADY`. Recall that your goal is to write a program that interacts and plays a series of games with `RANDY`. Suppose `myprogram` is your executable program, the following command starts the interactions:

```
./interact.sh ./myprogram ./RANDY n
```

where n is the number of games (set to 5,000 during evaluation).

By issuing the above command, messages displayed by `myprogram` will be read by `RANDY`, and messages displayed by `RANDY` will be read by `myprogram`. In other words, the standard output of `myprogram` gets sent as standard input of `RANDY` and vice versa. Since the standard output is "redirected" and not longer sent to the monitor, no message will be displayed on the monitor.

In your home directory, you can find sample C, C++, and Pascal programs, showing how your program can interact with `RANDY`. It is important to note that, in order to interact properly, your program needs to *flush* the standard output after every write operation. See the sample programs for details of flushing. Also note that your program must loop and play the game 5,000 times. Do not submit a program that plays only a single game.



Scoring

RANDY will play 5000 games with your submitted program. Your score will be derived from the number of games you win.

number games won	ratio to total number of games	score
≥ 2750	$\geq 55\%$	4
≥ 3150	$\geq 63\%$	8
≥ 3500	$\geq 70\%$	12
≥ 3700	$\geq 74\%$	16
≥ 3800	$\geq 76\%$	20

Note that the expected percentage of wins made by an optimal strategy is more than 77%.

During scoring, the random moves made by RANDY and the strength tables are not the same as those generated by the program RANDY passed to you. The strength table for the first game is always the one shown in the above example. Subsequent strength tables will be randomly generated.

Programming tips

1. By issuing `interact.sh`, messages sent to the standard output will not be displayed on the monitor. For debugging, you may send messages to the "standard error". Those messages will be displayed. For example,

```
Pascal:   Writeln(stderr, 'hello');
C:        fprintf (stderr, "hello");
C++:      cerr << "hello" << endl;
```

However, keep in mind that in the final submitted version, you **must not** write messages to the standard error.

2. It is helpful to note that the program RANDY accepts optional second and third parameter.

./RANDY n s v

As mentioned before, n is the number of games. The parameter s is the random seed. By changing the random seed, the moves made by RANDY will be different. When v is 1, the verbose mode is turned on and RANDY will send all messages exchanged and some useful information to standard error. When v is 2, only a single message on the number of games you have won will be sent to standard error. For example, the following uses 105 as random seed and does not turn on the verbose mode,

```
./RANDY  1 105
```

The following command turns on the verbose mode.

```
./interact.sh ./myprogram ./RANDY  1 105 1
```