Rock, paper, and scissors

This is a game everyone has played. There are two players in the game. Each player makes a choice of a hand shape, rock, paper, or scissors and compares their choices.

In this problem, we will use two threads to simulate two players playing the game for `n` rounds. A third thread serves as a referee, a trusted party that compares players' choices. In each round, plays make a decision and inform the referee their choice. The referee compares the choices and annouces the result to both players.

`thread_player()` and `thread_referee()` are the main function of two types of thread. Compelete the functions so two players can play the game.

Structure `shared_int_t` is used for players to send their choice to referee. The referee shares `choice1` with player 1 and `choice2` with player 2. `mutex` and `cond` are used for synchronization.

Sturcture `result_t` is for referee to announce the outcome to players. The referee shares `result` with both players. `barrier` is used for synchronization.

The program takes several arguments. For example, `-n` option specifies the number of rounds. Read the code for details.

Here is the sample output of the program.

```
 $ ./rock-paper -q
Player 1 won   2 times, lost   5 times, and tied   3 times.
Player 2 won   5 times, lost   2 times, and tied   3 times.

$ ./rock-paper -n100 -q
Player 1 won  29 times, lost  38 times, and tied  33 times.
Player 2 won  38 times, lost  29 times, and tied  33 times.

$ ./rock-paper -n1000 -q
Player 1 won 318 times, lost 333 times, and tied 349 times.
Player 2 won 333 times, lost 318 times, and tied 349 times.
```

Additonal challenges

Try different synchronization mechanisms.

Use mutex and cond in `result_t`

Sturcture `result_t` does not have to use barrier for synchronization. It can rely on mutex and cond.

Use barrier in `shared_int_t`

In this problem, we can use barrier in `shared_int_t` for synchronization.