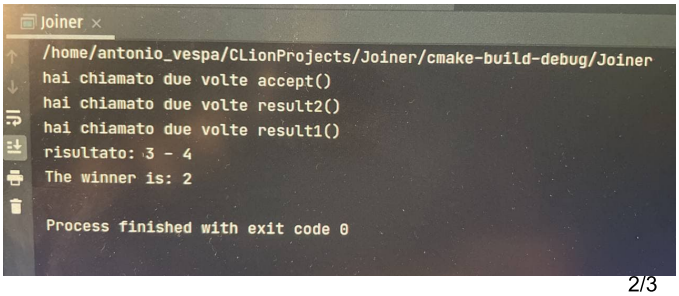
Implementare classe Challenge che aveva, oltre al costruttore, i seguenti metodi:
-void accept(Boolean response)
Indica il responso dello sfidato alla richiesta di giocare
-void result1(int score)
Indica lo score dello sfidante
-void result2(int score)
Indica lo score dello sfidato
-int winner()
Ritorna:
-0 se è pari
-1 se ha vinto lo sfidante
-2 se ha vinto lo sfidato
- -1 se la richiesta è stata rifiutata
Accept deve precedere result1 e result2, che invece non hanno ordine di chiamata tra loro, e tutti e 3 possono essere chiamati una sola volta, la violazione dei vincoli genera eccezione.
Winner si mette in attesa senza consumare CPU se il gioco è in corso o non è ancora cominciato

```cpp
#include <iostream>
#include <map>
#include <thread>
#include <condition_variable>
#include <mutex>
#include <chrono>
using namespace std::chrono_literals;

class Challenge {
private:
    bool _accepted, _started, _isPresS1, _isPresS2; //true,false
    int _winner; //winner 0,1,2, -1 non ancora deciso
    int _score1, _score2;
    std::mutex _m;
    std::condition_variable _cv;

public:
    Challenge(): _accepted(false), _started(false), _winner(-1), _isPresS1(false), _isPresS2(false){};
   void accept(bool response){
       std::unique_lock ul(_m);
       if(_started)
           throw  std::exception();
       _accepted=response;
       _started=true;
       if(!response) {
           _isPresS1 = true; //la partita è terminata se non ha accettato lo sfidato
           _isPresS2 = true;
       }
       _cv.notify_all();
   };

   void result1(int score){
       std::unique_lock ul(_m);
       _cv.wait(ul,[this](){return _started; });
       if(!_accepted)
           return;
       if(_isPresS1)
           throw  std::exception();
       _score1 = score;
       _isPresS1 = true;
       _cv.notify_all();
   };

   void result2(int score){
       std::unique_lock ul(_m);
       _cv.wait(ul,[this](){return _started; });
       if(!_accepted)
           return;
       if(_isPresS2)
           throw  std::exception();
       _score2 = score;
       _isPresS2 = true;
       _cv.notify_all();
   };

   int winner(){
       std::unique_lock ul(_m);
       _cv.wait(ul,[this](){return _isPresS2&&_isPresS1;}); //fine in ogni caso
       if(!_accepted)
```

```
60              return -1;
61          std::cout << "risultato: " << _score1 << " - " << _score2 << std::endl;
62          if(_score1<_score2)
63              _winner = 2;
64          else if(_score1>_score2)
65              _winner = 1;
66          else
67              _winner = 0;
68          switch(_winner){  //0=pari, 1=sfidante, 2=sfidato
69              case 0:
70                  return 0;
71              case 1:
72                  return 1;
73              case 2:
74                  return 2;
75              case -1:
76                  throw std::exception();
77          }
78          return 0;
79      };
80  };
81
82  int main() {
83      Challenge ch;
84
85      std::thread t1([&ch](){
86          //ch.accept(false);
87          std::this_thread::sleep_for(2000ms);
88          ch.accept(true);
89          try {
90              ch.accept(false);
91          }
92          catch(std::exception) {
93              std::cout << "hai chiamato due volte accept()"<< std::endl;
94          }
95          ch.result1(3);
96          try {
97              ch.result1(4);
98          }
99          catch(std::exception) {
100             std::cout << "hai chiamato due volte result1()"<< std::endl;
101         }
102     });
103
104     std::thread t2([&ch](){
105         ch.result2(4);
106         try {
107             ch.result2(4);
108         }
109         catch(std::exception) {
110             std::cout << "hai chiamato due volte result2()"<< std::endl;
111         }
112     });
113     int res = ch.winner();
114
115     std::cout << "The winner is: " << res <<
116
117     t1.join();
118     t2.join();
119     return 0;
```



```
120 }
121
```