```
class Executor {
public:
Executor(Context *ctx);
~Executor();
std::future<void> submit(std::function<void(Context*)> f);
void shutdown();
};
```

Quando viene costruita la classe, viene generato un unico thread che ha il compito di estrarre le funzioni dalla coda, immesse tramite il metodo submit(), ed eseguirle. Esso termina quando viene chiamato il metodo shutdown() o il distruttore della classe.
Le funzioni da svolgere vengono immesse tramite un metodo submit, che torna un oggetto future col risultato dell'elaborazione. Se il metodo viene chiamato quando è già stato chiamato il metodo shutdown, viene lanciata un'eccezione di tipo std::logic_error.
Il metodo shutdown viene chiamato per terminare l'esecuzione, e se ci sono ancora delle funzioni in coda, esse vengono comunque svolte.
Implementare la classe blabla

```cpp
#include <iostream>
#include <future>
#include <thread>
#include <functional>
#include <queue>


class Context{};

class Executor {
    std::queue<std::packaged_task<void(Context *)>> Coda;
    std::mutex m;
    std::condition_variable cv;
    std::thread t;
    bool _shutdown;
    Context *ctx;

public:
    Executor(Context *ctx) : _shutdown(false), ctx(ctx) {
        // Devo far avviare un thread
        t = std::thread([this]() {
            //Il thread deve girare sempre a meno che non viene chiamata
            // Shutdown
            while (true) {
                std::unique_lock<std::mutex> ul(m);
                //Il thread resta in attesa se non ha valori nella cosa
                //Si attiva se viene chiamata shutdown.
                cv.wait(ul, [this]() { return (!Coda.empty() || (this->_shutdown == true && Coda.empty())); });
                if (this->_shutdown && Coda.empty())
                    break;  // Esco
                auto x = std::move(Coda.front()); // Prendo la funzione
                Coda.pop(); //Tolgo la funzione
                x(this->ctx);  // Eseguo la funzione.
            }
        });
    }

    ~Executor(){
        std::lock_guard<std::mutex> l(m);
        if(_shutdown == false)
        {
            //Non è stato chiamato shutdown
            _shutdown = true;
            cv.notify_one();
        }
        t.join(); //chiave esercizio!!!
    }

    std::future<void> submit(std::function<void(Context *)> f) {
        //Devo inserire la funzione nella coda
        std::unique_lock<std::mutex> ul(m) ;
        if(_shutdown) // E' stato chiamato dopo aver fatto shutdown
            throw std::logic_error("Err");
        std::packaged_task<void(Context*)> x(f);
        std::future<void> Future = x.get_future() ;
        Coda.push(std::move(x));
        cv.notify_one() ; // Sveglio il thread se la coda era vuota.
        return Future ;
    }
```

```
60
61      void Shutdown()
62      {
63          std::unique_lock<std::mutex> ul (m);
64          _shutdown= true;
65          cv.notify_one();  // Sveglio il thread, faccio eseguire le ultime funzioni
66      }
67  };
68
69
```