Timer library

C++

```cpp
# ifndef TIMER_HPP
# define TIMER_HPP

# include <iostream>
# include <chrono>
# include <vector>

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * USAGE: Timer t;                                              *
 *        t.start();                                            *
 *        for (bla) { ... stuff happening ...; t.lap(); }       *
 *        double min = t.min(),                                 *
 *               mean = t.mean();                               *
 *                                                              *
 *        OR                                                    *
 *                                                              *
 *        Timer t;                                              *
 *        t.start();                                            *
 *        ...  stuff happening ...                              *
 *        t.stop();                                             *
 *                                                              *
 *        NOTE: stop() and lap() are equivalent!                *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

class Timer {
  typedef std::chrono::nanoseconds prec;
  typedef std::chrono::high_resolution_clock clock;
  typedef std::chrono::duration<double> duration_t;
  private:
    static const unsigned divisor = 1e9;
    clock::time_point t_start, t_end;
    duration_t t_min;
    std::vector<duration_t> t_laps;

  public:

   Timer();
   void start();
   void stop();
   void lap();
   void reset();

   double duration() const;
   double mean() const;
   double min() const;

};

// start the timer
void Timer::start(){
  t_start = clock::now();
  t_end = t_start;
}
```

```cpp

// stop the timer (equivalent to lap)
void Timer::stop(){
  // stop is just another lap
  lap();
}

// new lap
void Timer::lap(){
  clock::time_point tmp = clock::now();

  // get laptime
  duration_t laptime;
  laptime = tmp - t_end;

  // check if this lap was faster
  if (t_min > laptime || t_laps.size() == 0) {
    t_min = laptime;
  }

  // save time of this lap
  t_laps.push_back(laptime);

  // save total time
  t_end = tmp;
}

// idle constructor
Timer::Timer() {}

// resets all values
void Timer::reset(){
  t_laps = std::vector<duration_t>();
  start();
}

// returns total duration timer has been running
double Timer::duration() const {
  if (t_laps.size() > 0){
    // returning time in seconds! thats what the divisor is for
    auto dur = std::chrono::duration_cast<prec>(t_end - t_start);
    return double(dur.count())/divisor;
  }
  else {
    std::cerr << "Before calling Timer::duration() you need to call Timer::lap() or Timer::stop()!\n";
    return 0.;
  }
}

// returns mean of all laps
double Timer::mean() const {
  if (t_laps.size() > 0){
    // save total time in std::chrono units
    auto total_time = t_laps[0];
    for (unsigned int i = 1; i < t_laps.size(); ++i) {
      total_time += t_laps[i];
    }
```

```
    // convert time to double
    auto total_dur = std::chrono::duration_cast<prec>(total_time);
    double avg = double(total_dur.count())/divisor;
    return avg;
  }
  else {
    std::cerr << "Before calling Timer::mean() you need to call Timer::lap() or Timer::stop()!\n";
    return 0.;
  }
}

// returns minimum of all laps
double Timer::min() const {
  auto min = std::chrono::duration_cast<prec>(t_min);
  return double(min.count())/divisor;
}

# endif
```