

Part II : Thread pool

1. Design a thread pool class with following features:
 - A. Allow users to send jobs into the pool
 - B. Allow any kind of callable objects as jobs
 - C. Maintain a job queue to store unfinished jobs
 - i. Hint: element type: `std::function`/`std::bind` or `package_task`
 - D. Have 5 threads always waiting for new jobs. Each thread will keep a record of total running time throughout the lifespan of the thread.
 - E. Threads are terminated(joined) only when the thread pool is destructed. The total running time of each thread will be shown on the screen upon destruction along with the `std::thread::id`.
 - F. Use condition variable and mutex to notify threads to do works
2. Write one function (named `print_1`), which can generate a random integer number and then print out '1' if the number is an odd number otherwise '0'. Note that `cout` is also a shared resource.
3. Write a `print_2` functor, which simply prints "2" on the screen. Use conditional variable to ensure that `print_2` functor can only be executed when there is no more `print_1` job to be executed.
4. In main, first send 496 functions and then 4 functors into the pool.

```

#include<queue>
#include<functional>
#include<iostream>
Void add()
{
    std::cerr<<"1"<<std::endl;
}

struct ADD
{
    void operator>()()
    {
        std::cerr<<"2"<<std::endl;
    }
};

int main(void)
{
    ADD a;
    std::queue< std::function<void(void)> > jobs;
    jobs.push( std::bind(add) );
    jobs.push( std::bind( std::bind(a) ) );
    jobs.push( std::bind(a) );

    while(!jobs.empty() )
    {
        jobs.front()();
        jobs.pop();
    }
    return 0;
}

```