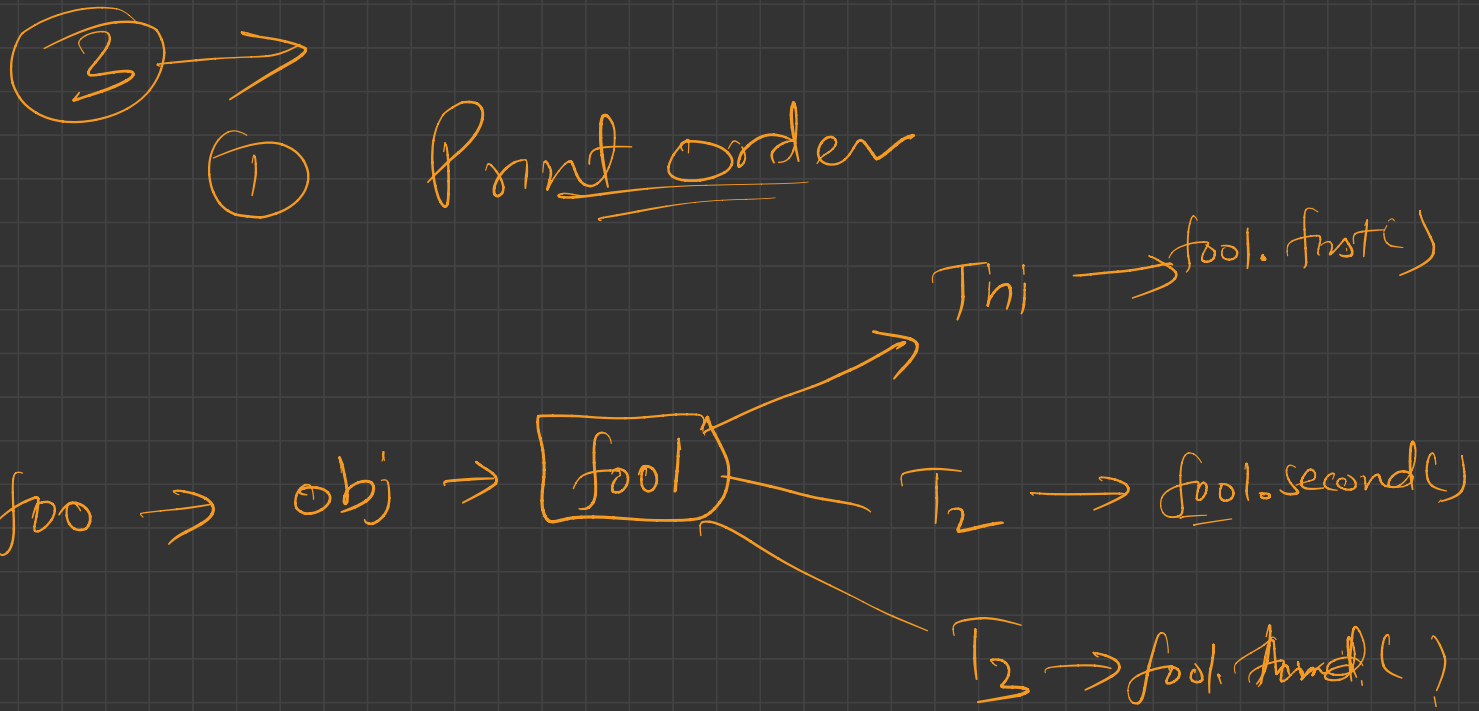
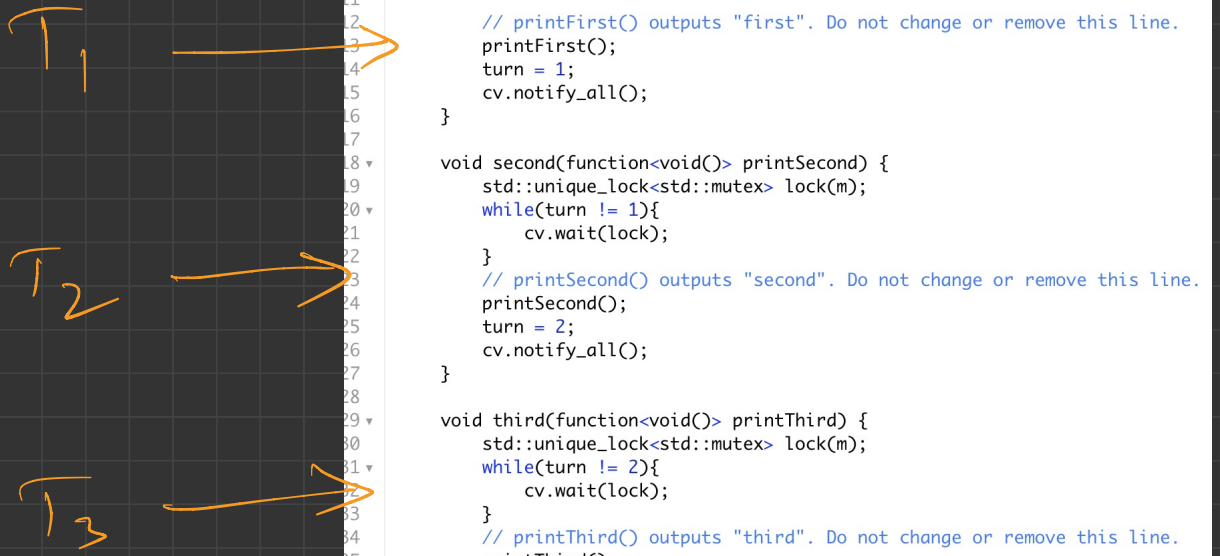
Lec-23


3
1
Print order
foo
obj
fool
Thi
fool.first()
T2
fool.second()
T3
fool.third()

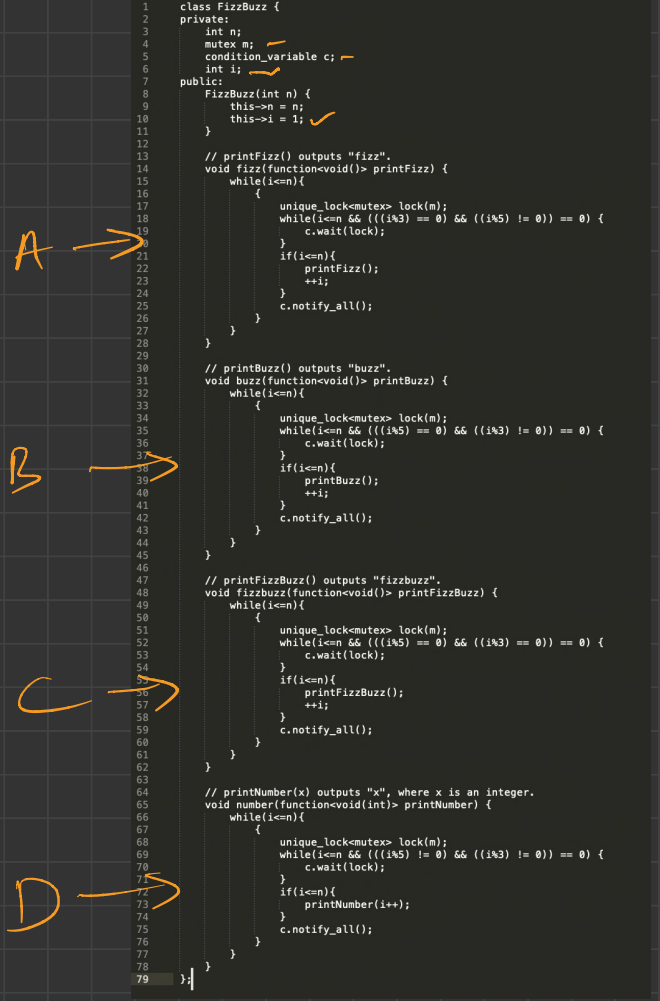
locks & conditional variable

1. first → printfirst() → T_1 execute.
2. when first is printed → T_2 execute.




```cpp
class Foo {
    std::mutex m;
    std::condition_variable cv;
    int turn;
public:
    Foo() {
        turn = 0;
    }

    void first(function<void()> printFirst) {

        // printFirst() outputs "first". Do not change or remove this line.
        printFirst();
        turn = 1;
        cv.notify_all();
    }

    void second(function<void()> printSecond) {
        std::unique_lock<std::mutex> lock(m);
        while(turn != 1){
            cv.wait(lock);
        }
        // printSecond() outputs "second". Do not change or remove this line.
        printSecond();
        turn = 2;
        cv.notify_all();
    }

    void third(function<void()> printThird) {
        std::unique_lock<std::mutex> lock(m);
        while(turn != 2){
            cv.wait(lock);
        }
        // printThird() outputs "third". Do not change or remove this line.
        printThird();
    }
};
```

(2) → Fizz buzz problem



```
class FizzBuzz {
private:
    int n;
    mutex m;
    condition_variable c;
    int i;
public:
    FizzBuzz(int n) {
        this->n = n;
        this->i = 1;
    }

    // printFizz() outputs "fizz".
    void fizz(function<void()> printFizz) {
        while(i<=n){
            {
                unique_lock<mutex> lock(m);
                while(i<=n && (((i%3) == 0) && ((i%5) != 0)) == 0) {
                    c.wait(lock);
                }
                if(i<=n){
                    printFizz();
                    ++i;
                }
                c.notify_all();
            }
        }
    }

    // printBuzz() outputs "buzz".
    void buzz(function<void()> printBuzz) {
        while(i<=n){
            {
                unique_lock<mutex> lock(m);
                while(i<=n && (((i%5) == 0) && ((i%3) != 0)) == 0) {
                    c.wait(lock);
                }
                if(i<=n){
                    printBuzz();
                    ++i;
                }
                c.notify_all();
            }
        }
    }

    // printFizzBuzz() outputs "fizzbuzz".
    void fizzbuzz(function<void()> printFizzBuzz) {
        while(i<=n){
            {
                unique_lock<mutex> lock(m);
                while(i<=n && (((i%5) == 0) && ((i%3) == 0)) == 0) {
                    c.wait(lock);
                }
                if(i<=n){
                    printFizzBuzz();
                    ++i;
                }
                c.notify_all();
            }
        }
    }

    // printNumber(x) outputs "x", where x is an integer.
    void number(function<void(int)> printNumber) {
        while(i<=n){
            {
                unique_lock<mutex> lock(m);
                while(i<=n && (((i%5) != 0) && ((i%3) != 0)) == 0) {
                    c.wait(lock);
                }
                if(i<=n){
                    printNumber(i++);
                }
                c.notify_all();
            }
        }
    }
};
```

③ Dining philosopher's problem

video no. 20

```cpp
class Semaphore
{
public:
    Semaphore() {}
    Semaphore(int c) : count(c){};

    void setCount(int a)
    {
        count = a;
    }

    inline void signal()
    {
        std::unique_lock<std::mutex> lock(mtx);
        count++;
        if(count <= 0)
            cv.notify_one();
    }

    inline void wait()
    {
        std::unique_lock<std::mutex> lock(mtx);
        count--;
        while (count < 0)
        {
            cv.wait(lock);
        }
    }

private:
    std::mutex mtx;
    std::condition_variable cv;
    int count;
};

class DiningPhilosophers {
    Semaphore fork[5];
    std::mutex m, l;
public:
    DiningPhilosophers() {
        for (int i = 0; i < 5; ++i) {
            fork[i].setCount(1);
        }
    }

    void wantsToEat(int philosopher,
                    function<void()> pickLeftFork,
                    function<void()> pickRightFork,
                    function<void()> eat,
                    function<void()> putLeftFork,
                    function<void()> putRightFork) {
        {
            std::lock_guard<std::mutex>lock(m);
            fork[(philosopher + 1) % 5].wait();
            fork[philosopher].wait();
        }
        pickLeftFork();
        pickRightFork();

        eat();

        putLeftFork();
        fork[(philosopher + 1) % 5].signal();
        putRightFork();
        fork[philosopher].signal();
    }
};
```