# 题目

编写一个可以从 1 到 n 输出代表这个数字的字符串的程序，但是：

如果这个数字可以被 3 整除，输出 "fizz"。

如果这个数字可以被 5 整除，输出 "buzz"。

如果这个数字可以同时被 3 和 5 整除，输出 "fizzbuzz"。

例如，当 n = 15，输出： 1, 2, fizz, 4, buzz, fizz, 7, 8, fizz, buzz, 11, fizz, 13, 14, fizzbuzz。

假设有这么一个类：

class FizzBuzz {

  public FizzBuzz(int n) { ... }  // constructor

public void fizz(printFizz) { ... } // only output "fizz"

public void buzz(printBuzz) { ... } // only output "buzz"

public void fizzbuzz(printFizzBuzz) { ... } // only output "fizzbuzz"

public void number(printNumber) { ... } // only output the numbers

}

请你实现一个有四个线程的多线程版  FizzBuzz， 同一个 FizzBuzz 实例会被如下四个线程使用：

线程A将调用 fizz() 来判断是否能被 3 整除，如果可以，则输出 fizz。

线程B将调用 buzz() 来判断是否能被 5 整除，如果可以，则输出 buzz。

线程C将调用 fizzbuzz() 来判断是否同时能被 3 和 5 整除，如果可以，则输出 fizzbuzz。

线程D将调用 number() 来实现输出既不能被 3 整除也不能被 5 整除的数字。

提示：

本题已经提供了打印字符串的相关方法，如 printFizz() 等，具体方法名请参考答题模板中的注释部分。

# 分析

## 方法一：互斥锁+条件变量

还是使用condition\_variable，注意要让所有的线程在执行完之后顺利推出。否则会时间超限。

class FizzBuzz {

private:

int n;

int num;

condition\_variable cv;

mutex m;

public:

FizzBuzz(int n) {

this->n = n;

num = 1;

}

// printFizz() outputs "fizz".

void fizz(function<void()> printFizz) {

unique\_lock<mutex> lk(m);

for ( ; num <= n ; )

{

if (num % 3 == 0 && num % 5 != 0)

{

printFizz();

++num;

cv.notify\_all();

}

cv.wait(lk, [this] {return (num > n) || num % 3 == 0 && num % 5 != 0; });

}

}

// printBuzz() outputs "buzz".

void buzz(function<void()> printBuzz) {

unique\_lock<mutex> lk(m);

for (; num <= n;)

{

if (num % 5 == 0 && num % 3 != 0)

{

printBuzz();

++num;

cv.notify\_all();

}

cv.wait(lk, [this] {return (num > n) || num % 5 == 0 && num % 3 != 0; });

}

}

// printFizzBuzz() outputs "fizzbuzz".

void fizzbuzz(function<void()> printFizzBuzz) {

unique\_lock<mutex> lk(m);

for (; num <= n;)

{

if (num % 3 == 0 && num % 5 == 0)

{

printFizzBuzz();

++num;

cv.notify\_all();

}

cv.wait(lk, [this] {return (num > n) || num % 3 == 0 && num % 5 == 0; });

}

}

// printNumber(x) outputs "x", where x is an integer.

void number(function<void(int)> printNumber) {

unique\_lock<mutex> lk(m);

for (; num <= n;)

{

if (num % 3 != 0 && num % 5 != 0)

{

printNumber(num);

++num;

if (num > n)

{

cv.notify\_all();

break;

}

if (num % 3 != 0 && num % 5 != 0)

{

continue;

}

else

{

cv.notify\_all();

}

}

cv.wait(lk, [this] {return (num > n) || num % 3 != 0 && num % 5 != 0; });

}

}

};

## 方法二：信号量

#include <semaphore.h>

#include <functional>

#include <thread>

using namespace std;

class FizzBuzz {

private:

int n;

int cur;

sem\_t sem\_fizz;

sem\_t sem\_buzz;

sem\_t sem\_fizz\_buzz;

sem\_t sem\_num;

public:

FizzBuzz(int n) {

this->n = n;

cur = 0;

sem\_init(&sem\_fizz, 0, 0);

sem\_init(&sem\_buzz, 0, 0);

sem\_init(&sem\_fizz\_buzz, 0, 0);

sem\_init(&sem\_num, 0, 1);

}

// printFizz() outputs "fizz".

void fizz(function<void()> printFizz) {

while(cur <= n){

sem\_wait(&sem\_fizz);

if(cur > n) break;

printFizz();

sem\_post(&sem\_num);

}

}

// printBuzz() outputs "buzz".

void buzz(function<void()> printBuzz) {

while(cur <= n){

sem\_wait(&sem\_buzz);

if(cur > n) break;

printBuzz();

sem\_post(&sem\_num);

}

}

// printFizzBuzz() outputs "fizzbuzz".

void fizzbuzz(function<void()> printFizzBuzz) {

while(cur <= n){

sem\_wait(&sem\_fizz\_buzz);

if(cur > n) break;

printFizzBuzz();

sem\_post(&sem\_num);

}

}

// printNumber(x) outputs "x", where x is an integer.

void number(function<void(int)> printNumber) {

while(++cur <= n){

sem\_wait(&sem\_num);

if(cur % 3 == 0 && cur % 5 == 0){

sem\_post(&sem\_fizz\_buzz);

}else if(cur % 3 == 0){

sem\_post(&sem\_fizz);

}else if(cur % 5 == 0){

sem\_post(&sem\_buzz);

}else{

printNumber(cur);

sem\_post(&sem\_num);

}

}

// 以下三个post通过更新sem\_fizz等信号量，调动其他线程运行，进而结束所有线程

sem\_post(&sem\_fizz);

sem\_post(&sem\_buzz);

sem\_post(&sem\_fizz\_buzz);

}

};

## 方法三：原子操作