## Concurrence

1) Variables atomiques

Constatez l'effet du code suivant :

```
auto compteur = 0;

auto incremente_compteur(int n) {
    for (int i = 0; i < n; ++i)
        ++compteur;
}
```

En lançant 2 thread sur cette fonction :

```
auto t1 = std::thread{ incremente_compteur, 1000000 };
auto t2 = std::thread{ incremente_compteur, 1000000 };
t1.join();
t2.join();

std::cout << "compteur = " << compteur << std::endl;

assert(compteur == 2000000);
```

Includes cassert, thread

Et corrigez en utilisant :

```
std::atomic<int> compteur = 0;
```

Include atomic

2) Producteur et consommateur

Créer un thread pour lancer cette fonction de génération :

```
void generate_ints() {
    for (auto i : { 1,2,3, done }) {
        std::this_thread::sleep_for(std::chrono::seconds(2));
        {
            std::unique_lock<std::mutex> lock(mtx);
            q.push(i);
            notifie = true;
            std::cout << "push " << i << std::endl;
            cv.notify_one();
        }
    }
```

Créer un thread pour lancer cette fonction de consommation :

```
void print_ints() {
    auto i = 0;
    std::cout << " entree dans print" << std::endl;
    while (i != done) {
        auto lock = std::unique_lock<std::mutex>{ mtx };
        while(!notifie) cv.wait(lock);

        while (!q.empty()) {

            i = q.front();
            q.pop();

        }
        notifie = false;
        if (i != done) {
            std::cout << " recup de " << i << '\n';
        }
    }
}
```

Déclarer les variables gobales utilisées et terminer le main avec des join sur les 2 threads.