

- Solution

The solution to designing a ping pong game using the concepts of multithreading is explained in this lesson.

WE'LL COVER THE FOLLOWING ^

- Solution
- Explanation

Solution

```
// conditionVariablePingPong.cpp

#include <condition_variable>
#include <iostream>
#include <thread>
#include <atomic>

bool dataReady= false;

std::mutex mutex_;
std::condition_variable condVar1;
std::condition_variable condVar2;

std::atomic<int> counter{};
int COUNTLIMIT=50;

void setTrue(){

    while(counter <= COUNTLIMIT){

        std::unique_lock<std::mutex> lck(mutex_);
        condVar1.wait(lck, []{return dataReady == false;});
        dataReady= true;
        ++counter;
        std::cout << dataReady << std::endl;
        condVar2.notify_one();

    }
}

void setFalse(){

    while(counter < COUNTLIMIT){

        std::unique_lock<std::mutex> lck(mutex_);
```

```

        std::unique_lock<std::mutex> lock(mutex_);
        condVar2.wait(lock, []{return dataReady == true;});
        dataReady= false;
        std::cout << dataReady << std::endl;
        condVar1.notify_one();

    }

}

int main(){

    std::cout << std::boolalpha << std::endl;

    std::cout << "Begin: " << dataReady << std::endl;

    std::thread t1(setTrue);
    std::thread t2(setFalse);

    t1.join();
    t2.join();

    dataReady= false;
    std::cout << "End: " << dataReady << std::endl;

    std::cout << std::endl;
}

```



Explanation

- Thread `t1` sets the `dataReady` variable to `true`, increments the `counter`, and sends its notification.
- Thread `t2` waits in line 25 for the notification. It sets `dataReady` to `false` and notifies thread `t1`.
- The game ends after 50 iterations.

For further information, read [std::condition_variable](#)

In the next lesson, we will see how multithreading in C++ becomes easier with tasks.