- Example

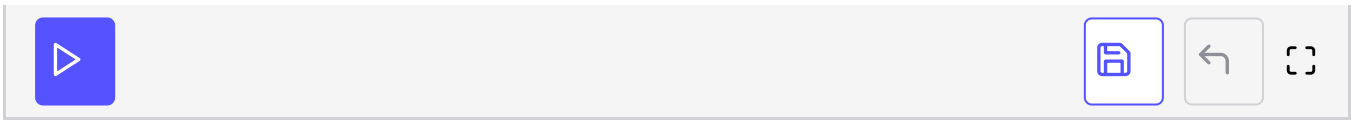
In this lesson, we will go over an example of condition variables.

WE'LL COVER THE FOLLOWING

Example

```
// conditionVariable.cpp

#include <iostream>
#include <condition_variable>
#include <mutex>
#include <thread>

std::mutex mutex_;
std::condition_variable condVar;

bool dataReady{false};

void doTheWork(){
  std::cout << "Processing shared data." << std::endl;
}

void waitingForWork(){
    std::cout << "Worker: Waiting for work." << std::endl;
    std::unique_lock<std::mutex> lck(mutex_);
    condVar.wait(lck, []{ return dataReady; });
    doTheWork();
    std::cout << "Work done." << std::endl;
}

void setDataReady(){
    {
      std::lock_guard<std::mutex> lck(mutex_);
      dataReady = true;
    }
    std::cout << "Sender: Data is ready."  << std::endl;
    condVar.notify_one();
}
```

```
int main(){

  std::cout << std::endl;


  std::thread t1(waitingForWork);
  std::thread t2(setDataReady);

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

  std::cout << std::endl;

}
```



Explanation

- The program has two child threads: `t1` and `t2`.
- They get their work packages, `waitingForWork` and `setDataReady`, in lines 38 and 39. Using the condition variable `condVar`, `setDataReady` notifies that its work is done with the preparation of the work: `condVar.notify_one()`.
- While holding the lock, thread `t1` waits for its notification: `condVar.wait(lck,[]{ return dataReady; })`.
- The sender and receiver need a lock. In the case of the sender, a `std::lock_guard` is sufficient, because it calls lock and unlock only once.
- In the case of the receiver, a `std::unique_lock` is necessary, because it frequently locks and unlocks its mutex.

The waiting thread has a quite complicated workflow.

The `wait` Workflow

If it is the first time `wait` is invoked, the following steps will occur.

- The call to `wait` locks the mutex and checks if the predicate `[]{ return dataReady; }` evaluates to true.
    - If true, the condition variable continues and unlocks the mutex at the end of its scope.

    - If false, the condition variable unlocks the mutex and puts itself back in the wait state.

Subsequent `wait` calls behave differently.

- The waiting thread gets a notification. It locks the mutex and checks if the predicate `[]{ return dataReady; }` evaluates to true.
    - If true, the condition variable continues and unlocks the mutex at the end of its scope.
    - If false, the condition variable unlocks the mutex and puts itself back in the wait state.

Test your knowledge on condition variables with an exercise in the next lesson.