

Issues of Mutexes: Avoiding Exceptions

This lesson lists some caveats pertaining to mutexes and how to avoid them in C++.

WE'LL COVER THE FOLLOWING



- Exceptions and Unknown Behavior
- Issues & Best practices

Exceptions and Unknown Behavior

The small code snippet has a lot of issues to look at, including a few exceptions and unknown behaviors in the program:

```
std::mutex m;  
m.lock();  
sharedVariable = getVar();  
m.unlock();
```



Issues & Best practices

1. If the function `getVar()` throws an exception, the mutex `m` will not be released.
2. Never ever call an unknown function while holding a lock. If the function `getVar` tries to lock the mutex `m`, the program has undefined behavior because `m` is not a recursive mutex. Most of the time, the undefined behavior will result in a deadlock.
3. Avoid calling a function while holding a lock. Maybe the function is from a library and you get a new version of the library, or the function is rewritten, but there is always the danger of a deadlock.

The more locks your program needs, the more challenging it becomes. The dependency is very non-linear.

