**Mutlithreading:**

We should not forge to call join or detach in case of exception, So to prevent this we use Resouce Acqusition Is Initialization (RAII).

In multithreading data sharing between objects is very easy, But easy sharing can cause problem. So one such problem is race condition.

RACE CONDITION:

It is a kind of bug that appears in multithreading.

When two or more threads access a same memory location parallely and one of the thread modifies the memory location, this can lead to unexpected results some times. This is called race condition.

Race condition are hard to find and reproduce as they dont occur every time.

To fix race condition in multi threaded environment we need mutex. i.e each thread needs to lock a mutex before modifying or reading the shared data and after modifying / reading the thread it should unlock the mutex.

**std::mutex**

mutex are <mutex> header file. The class representing a mutex is std::mutex class.

There are two important methods of mutex:

> lock()

> unlock()

We lock the mutex before accessing the shared data which has to be modified and then unlock it after necessary operations are done.

What happens is we forgot to ulock it, then others cannot access it. This kind of scenario can happen in some kind of scenario.

**std::lock\_guard**

std::lock\_guard is a class template, it implements the Resourse Acquisition Is Initialization (RAII) for mutex. It wraps the mutex inside its object and locks the attached mutex in its constructor. When destructor is called it releases the mutex.

Example:

std::lock\_guard<std::mutex> lockGuardName( mutexName);

Lock guard usage is easier than mutex as we dont need to mention to unlock it.

**Sleep for a duration**

We can block a thread for a specified dureation by using the function,

syntax:

std::this\_thread::sleep\_for(std::chrono::duration)

std::chrono::nanoseconds(value)

std::chrono::microseconds(value)

std::chrono::milliseconds(value)

std::chrono::seconds(value)

std::chrono::hours(value)

**Sleep until a timepoint**

Making a thread to sleep until a timepoint in future.