Thread-Safe Initialization: Constant Expressions

This lesson gives an overview of thread-safe initialization from the perspective of concurrency in C++ with Constant Expressions.

Constant expressions are expressions that the compiler can evaluate at compile-time; they are implicitly thread-safe. Using the keyword `constexpr` in with a variable makes the variable a constant expression. The constant expression must be initialized immediately.

```
constexpr double pi = 3.14;
```

Additionally, user-defined types can also be constant expressions. For those types, there are a few requirements that must be met in order to initialize it at compile-time.

- They must not have virtual methods or a virtual base class.
- Their constructor must be empty and itself be a constant expression.
- Their methods, which should be callable at compile-time, must be constant expressions.

Instances of `MyDouble` satisfy all these requirements, so it is possible to instantiate them at compile-time. This instantiation is thread-safe.

```
// constexpr.cpp

#include <iostream>

class MyDouble{
  private:
    double myVal1;
    double myVal2;
  public:
    constexpr MyDouble(double v1,double v2):myVal1(v1),myVal2(v2){}
    constexpr double getSum() const { return myVal1 + myVal2; }
};

int main() {
  constexpr double myStatVal = 2.0;
  constexpr MyDouble myStatic(10.5, myStatVal);
  constexpr double sumStat= myStatic.getSum();

  std::cout << "SumStat: "<<sumStat << std::endl;
}
```

In the next lesson, we'll discuss thread-safe initialization from the perspective of concurrency in C++ with `call_once` and `once_flag`.