A `LockableStream` holds a reference to an `std::ostream` object and an `std::mutex`. The streaming operator, **`operator<<`**, constructs a `LockedStream` object and delegates the actual streaming operation to the `LockedStream`. The return value of **`operator<<`** is an *rvalue* of type `LockedStream`. We can then create a `LockableStream`, supplying it only with the wrapped `std::ostream`, and print to it:

```
LockableStream lockableCout(std::cout);

void f2()
{
    lockableCout << "Hello, " << 2021 << '\n';
}
```

Optimizing immutable types and builder classes

An **immutable type** is a type that has no modifying operations. Among other benefits, the representation of an immutable type can be shared by all objects that have the same value, including in concurrent threads. Every object that logically "modifies" an object of immutable type does so by returning a new object having the modified value; the original object remains unchanged. An `ImmutableString` class, for example, might have an `insert` member function that takes a second string argument and returns a copy of the original string with the second string inserted in the specified location:

```
#include <memory>    // std::shared_ptr
#include <string>    // std::string
#include <iostream>  // std::ostream, std::cout, std::endl

class ImmutableString
{
    std::shared_ptr<std::string> d_dataPtr;

    static const std::string s_emptyString;

public:
    using size_type = std::string::size_type;

    ImmutableString() {}

    ImmutableString(const char* s)
        : d_dataPtr(std::make_shared<std::string>(s)) { }

    ImmutableString(std::string s)
        : d_dataPtr(std::make_shared<std::string>(std::move(s))) { }

    ImmutableString& operator=(const ImmutableString&) = delete;
```