```
class RedString
{
   std::string d_value;

public:
    RedString(const char* s = "") : d_value("Red: ") { d_value += s; }

          std::string&  value() &       { return d_value; }
    const std::string&  value() const & { return d_value; }
          std::string&& value() &&      { return std::move(d_value); }
    // Note that this third overload returns std::string by rvalue reference.

    // ...
};

void f1()
{
          RedString urs("hello");
    const RedString crs("world");

    std::string h1 = urs.value();                   // "Red: hello"
    std::string h2 = crs.value();                   // "Red: world"
    std::string h3 = RedString("goodbye").value();  // "Red: goodbye"

    std::string h4 = std::move(urs).value();        // "Red: hello"
    std::string h5 = urs.value();                   // Bug, unspecified value

    std::string h6 = std::move(crs).value();        // "Red: world"
    std::string h7 = crs.value();                   // OK, "Red: world"
}
```

The `RedString` class provides three ref-qualified overloads of `value`. When `value` is called on `urs` and `crs`, the non**`const`** and **`const`** *lvalue*-ref-qualified overloads, respectively, are selected. Both overloads return an lvalue reference to `std::string`, so `h1` and `h2` are constructed using the **copy constructor**, as usual. In the case of the **temporary** ~~variable~~ created by `RedString("goodbye")`, however, the *rvalue*-ref-qualified overload of `value` is selected. This overload returns an rvalue reference, so `h3` is constructed using the **move constructor**, which might be more efficient.

As in the case of most such code, it is assumed that an rvalue reference refers to an object whose state no longer matters after evaluation of the expression. When that assumption doesn't hold, unexpected results may occur, as in the case of `h5`, which is initialized from a moved-from string, yielding a **valid but unspecified** string **value**.

The `value` member function is not overloaded for a **`const`** *rvalue*-ref-qualified object. Invoking it for such a (rarely encountered) type selects the **`const`** *lvalue*-ref-qualified overload, as *rvalues* can always be bound to **`const`** lvalue references. As a result, `h6` is initialized from a **`const`** `std::string&`, invoking the copy constructor and leaving `crs` unmodified.