Item 6: Temporary Objects

Le Van Huynh

October 22, 2023

Contents

1 Question **1**

2 Answer **1**

1 Question

Unnecessary and/or temporary objects are frequent culprits that can throw all your hard work—and your program's performance—right out the window. How can you spot them and avoid them?

You are doing a code review. A programmer has written the following function, which uses unnecessary temporary objects in at least three places. How many can you identify, and how should the programmer fix them?

```
string FindAddr(list<Employee> emps, string name)
{
    for (auto i = emps.begin(); i != emps.end(); i++) {
        if (*i == name) {
            return i->addr;
        }
    }
    return "";
}
```

2 Answer

The function parameters can be passed as `const` references:

```
string FindAddr(const list<Employee>& emps,
                const string& name);
```

`i++` creates a temporary, so `++i` may be more efficient.

`emps.end()` is re-calculated every time. To avoid both this and the previous issue, we can use the for-each syntax:

```
for (const auto &e: emps) {
    // ...
}
```

From the code, the `Employee` class has a comparison operator or a conversion to `string`. The first case may introduce a temporary object depending on how it is implemented, and the second case always creates a temporary `string`.

Finally, the function creates a string to return, but that cannot be avoided.

I would fix the function as follows:

```
string FindAddr(const list<Employee>& emps,
                const string& name)
{
    for (const auto& e: emps) {
        if (e.getName() == name) {
            return e.addr;
        }
    }
    return "";
}
```

Or use the built-in `find` function:

```
string FindAddr(const list<Employee>& emps,
                const string& name)
{
    auto it = std::find_if(emps.begin(), emps.end(),
                           [&name](const auto& e) {
                               return e.getName() == name;
                           });
    return (it == emps.end()) ? "" : it->addr;
}
```