C++14
and CLAS12 Software

Johann Goetz
June, 2013

Wait, why C++14 instead of C++11!?

I am following Herb Sutter's lead on this:

> "C++14 is going to be current C++ soon. As of April, it's now feature-complete, and the detailed technical specification should be frozen this fall or winter. Compilers and libraries are already aggressively implementing it. I wouldn't be surprised if we had multiple fully-conforming C++14 implementations by the end of next year... [and] **C++14 is a small delta**."
>
> -- from Guru of the Week by Herb Sutter

Wait, why C++14 instead of C++11!?

I am following Herb Sutter's lead on this:

> However, I will give no C++14 specific examples. I will not give examples for features not implemented in gcc version 4.7
>
> Also, I suspect that very soon the "14" will be dropped and when someone says "C++" the "14" will be assumed (just as "98" is assumed today).

Move semantics and smart pointers

When returning a large object from a function, there are many options:

1. return by value (easiest, "safe" and clean, but expensive)

```
class LargeObj;

LargeObj fn()
{
    LargeObj obj;
    // returns a copy
    return obj;
}
```

Move semantics and smart pointers

When returning a large object from a function, there are many options:

2. return pointer to a new'd object (who owns this object? That is, who is in charge of deleting it?)

```
class LargeObj;

LargeObj* fn()
{
    LargeObj* obj = new LargeObj();
    // exception thrown => memory leak!
    return obj;
}
```

Move semantics and smart pointers

When returning a large object from a function, there are many options:

3. smart pointer to rescue? Safer, but deletion still an issue.

```
class LargeObj;

LargeObj* fn()
{
    unique_ptr<LargeObj> obj{new LargeObj};
    // exception thrown => memory is cleaned up.
    return obj.release();
}
```

Move semantics and smart pointers

When returning a large object from a function, there are many options:

3. smart pointer to rescue? (can avoid raw pointers)

```
class LargeObj;

unique_ptr<LargeObj> fn()
{
    unique_ptr<LargeObj> obj{new LargeObj};
    // exception thrown => memory is cleaned up.
    // unique_ptr is moved (owner changes hands)
    return obj;
}
```

Move semantics and smart pointers

The Clara framework uses unique_ptr in all cases where you'd normally expect a raw pointer.

A note about unique_ptr - as opposed to shared_ptr:

> "**unique_ptr** is represented by a simple built-in pointer and the overhead of using one compared to a built-in pointer are miniscule. In particular, **unique_ptr** does not offer any form of dynamic checking."
>
> -- Bjarne Stroustrup (FAQ webpage)

Move semantics and smart pointers

When returning a large object from a function, there are many options:

4. Can we do better than unique_ptr?

Move semantics and smart pointers

When returning a large object from a function, there are many options:

4. Yes: Move it! Consider the following example:

```
class LargeObj;

LargeObj fn()
{
    LargeObj obj;
    // exception-safe, like unique_ptr sol'n
    // cast obj to LargeObj&& and "move" it out
    return std::move(obj);
}
```

Move semantics and smart pointers

When returning a large object from a function, there are many options:

4. Alternative to std::move is a Move Constructor

```
class LargeObj;

void LargeObj::LargeObj(LargeObj&& obj)
{
    *this = std::move(obj);
};

LargeObj fn()
{
    LargeObj obj;
    return obj;
}
```

A word about initializer lists, auto and range-based for-loops

Quick example. Consider the following:

```
vector<double> arr;

for (int i=1; i<6; i++) {
    arr.push_back(i);
}

for (vector<double>::iterator itr = arr.begin();
     itr != arr.end();
     ++itr) {
    cout << *itr << endl;
}
```

A word about initializer lists, auto and range-based for-loops

Quick example. These three features are ubiquitous in all of my code since January, 2013.

```
vector<double> arr {1,2,3,4,5};

for (auto x : arr)
{
    cout << x << endl;
}
```

References I use when coding

very lucid, small examples, almost feature complete
http://www.cplusplus.com/reference/

feature complete, but missing some advanced examples
http://en.cppreference.com/w/

Boost is almost essential for non-trivial libraries and programs
http://www.boost.org/doc/libs/?view=categorized

Fun reading for those that want deeper discussions about C++ style

The inventor's website has lots of detailed discussions about features you might never use, but probably should!

http://www.stroustrup.com

Herb Sutter's Guru of the Week has small examples of typical problems and their solutions. (Check out his associated blog!)

http://www.gotw.ca