Expressing memory management in C++

Marc-Olivier Andrez
Senior Software Engineer at AMIA-Systems

Example

```
1    #include <iostream>
2
3    struct B {
4        int val;
5    };
6
7    class A {
8    public:
9        A(B* b);
10       ~A();
11   private:
12       B* b;
13   };
14
15   int main() {
16       B* b = new B{42};
17       A a(b);
18       std::cout << "Hello, " << b->val << "!" << std::endl;
19       delete b;
20       return 0;
21   }
```

1. **Should I delete b?**
2. **Did A take ownership of b?**

Possible implementations

```
// A doesn't own b
// External object

A::A(B* b_)
   : b(b_)
{
}

A::~A() {

}
```

```
// A owns b
// Dependency Injection

A::A(B* b_)
   : b(b_)
{
}

A::~A() {
   delete b;
}
```

```
$ clang++ -std=c++1z -g -fsanitize=address 01-intro.cpp
$ ./a.out
==9422==ERROR: AddressSanitizer: attempting double-free on 0x60200000ef30 in
thread T0:
    #0 0x10e10c57b in wrap__ZdlPv (/usr/local/lib/clang/3.9.0/lib/darwin/
libclang_rt.asan_osx_dynamic.dylib+0x5857b)
    #1 0x10e0a8fea in A::~() ExpressingMemoryInCxx/code/01-intro.cpp:29:2
    #2 0x10e0a8f24 in A::~() ExpressingMemoryInCxx/code/01-intro.cpp:28:9
    #3 0x10e0a8ab8 in main ExpressingMemoryInCxx/code/01-intro.cpp:21:1
    #4 0x7fff98e38254 in start (/usr/lib/system/libdyld.dylib+0x5254)
    #5 0x0  (<unknown module>)

0x60200000ef30 is located 0 bytes inside of 4-byte region
[0x60200000ef30,0x60200000ef34)
freed by thread T0 here:
    #0 0x10e10c57b in wrap__ZdlPv (/usr/local/lib/clang/3.9.0/lib/darwin/
libclang_rt.asan_osx_dynamic.dylib+0x5857b)
    #1 0x10e0a8aa5 in main ExpressingMemoryInCxx/code/01-intro.cpp:19:2
    #2 0x7fff98e38254 in start (/usr/lib/system/libdyld.dylib+0x5254)
    #3 0x0  (<unknown module>)

previously allocated by thread T0 here:
    #0 0x10e10bfbb in wrap__Znwm (/usr/local/lib/clang/3.9.0/lib/darwin/
libclang_rt.asan_osx_dynamic.dylib+0x57fbb)
    #1 0x10e0a894d in main ExpressingMemoryInCxx/code/01-intro.cpp:16:9
    #2 0x7fff98e38254 in start (/usr/lib/system/libdyld.dylib+0x5254)
    #3 0x0  (<unknown module>)

SUMMARY: AddressSanitizer: double-free (/usr/local/lib/clang/3.9.0/lib/darwin/
libclang_rt.asan_osx_dynamic.dylib+0x5857b) in wrap__ZdlPv
==9422==ABORTING
Abort trap: 6
```

How to express memory ownership?

Using comments

```
class A {
public:
   A(B* b); // Takes ownership
   ~A();
private:
   B* b;
};
```

1. **We do not read comments!**
2. **They don’t appear when IDE makes code completions**

Documentation

QWidget::QWidget(QWidget *_parent_ = Q_NULLPTR, Qt::WindowFlags _f_ = Qt::WindowFlags())

Constructs a widget which is a child of *parent*, with widget flags set to *f*.

If *parent* is 0, the new widget becomes a window. If *parent* is another widget, this widget becomes a child window inside *parent*. The new widget is deleted when its *parent* is deleted.

http://doc.qt.io/qt-5/qwidget.html

"Writing Good C++14"

Bjarn Stroustrup, CppCon 2015

https://www.youtube.com/watch?v=1OEu9C51K2A

C++ Core Guidelines

- *The C++ Core Guidelines are a collaborative effort led by Bjarne Stroustrup, much like the C++ language itself. They are the result of many person-years of discussion and design across a number of organizations.*

- *The guidelines are focused on relatively higher-level issues, such as interfaces, resource management, memory management, and concurrency. Such rules affect application architecture and library design. Following the rules will lead to code that is statically type-safe, has no resource leaks, and catches many more programming logic errors than is common in code today. And it will run fast -- you can afford to do things right.*

- https://github.com/isocpp/CppCoreGuidelines

“Never transfer ownership by a raw pointer (T*)”

C++ Core Guidelines, item I.11

https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md#i11-never-transfer-ownership-by-a-raw-pointer-t

```cpp
#include <iostream>
#include <memory>

struct B {
    int val;
};

class A {
public:
    A(std::unique_ptr<B> b);
    ~A(); // Not required unless using forward declaration for B
    const B* getB() const {
        return b.get();
    };
private:
    std::unique_ptr<B> b;>><
};

int main() {
    std::unique_ptr<B> b(new B{42});
    A a(std::move(b));
    // Cannot use b anymore
    std::cout << "Hello, " << a.getB()->val << "!" << std::endl;
    return 0;
}

```

Thank you!