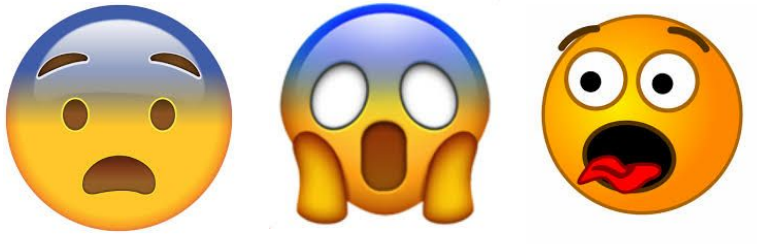
Slicing in the standard library



Amir Kirsh
Academic College of Tel-Aviv-Yaffo
and Tel-Aviv University
(kirshamir at gmail com)

Slicing

```
struct A {
    int _a;
    A(int a = 3): _a{a} {}
};

struct B: A // public inheritance
{
    int _b;
    B(int a = 3, int b = 3): A(a), _b{b} {}
};

A a = B{2, 2}; // ...chip chop... easy case - yes
```

Slicing

```
std::vector<A> vec;

B b;

vec.push_back(b); // chip chop?
```

Slicing

```
std::vector<A> vec;

B b;

vec.push_back(b); // chip chop? yes of course...
```

Slicing

```
void foo(A& a, bool bar) {
    if(bar) a = B{}; // chip chop?
}

int main() {
    B b1{2, 2};
    foo(b1, true);
}
```

Slicing

```
void foo(A& a, bool bar) {
    if(bar) a = B{}; // chip chop? yes, sort of
}

int main() {
    B b1{2, 2};
    foo(b1, true);
}
```

http://coliru.stacked-crooked.com/a/f52b2058cbe0a896

See also: https://www.learncpp.com/cpp-tutorial/12-8-object-slicing/
https://stackoverflow.com/questions/274626/what-is-object-slicing

Slicing in the standard library?

Slicing is considered a bad thing...

Yet it appears somewhere inside the standard library.

Any guess where?

Slicing in the standard library?

Slicing is considered a bad thing...

Yet it appears somewhere inside the standard library.

Any guess where?

Before we answer this question let’s go for a journey in the smart pointers domain

Polymorphism with Smart Pointers

wait, no slicing here yet

```
struct A {
    virtual ~A() { /*...*/ }
    // ...
};
```

```
struct B: A {
    ~B() { /*...*/ }
    // ...
};
```

```
int main() {
    // unique_ptr
    unique_ptr<A> uptr = make_unique<A>();
    uptr = make_unique<B>();
    // shared_ptr
    shared_ptr<A> sptr = make_shared<A>();
    sptr = make_shared<B>();
}
```

which dtors would be called?

http://coliru.stacked-crooked.com/a/5c22c5c9493f154f

Polymorphism with Smart Pointers

surprise!!

```
struct A {
    /* virtual */ ~A() { /*...*/ }
    // ...
};
```

```
struct B: A {
    ~B() { /*...*/ }
    // ...
};
```

```
int main() {
    // unique_ptr
    unique_ptr<A> uptr = make_unique<A>();
    uptr = make_unique<B>();
    // shared_ptr
    shared_ptr<A> sptr = make_shared<A>();
    sptr = make_shared<B>();
}
```

which dtors
would be called?

http://coliru.stacked-crooked.com/a/361c149a6f87d36f

shared_ptr vs. unique_ptr

shared_ptr vs. unique_ptr polymorphic behavior on destruction:

http://coliru.stacked-crooked.com/a/3347282d011b8e00

they do not behave the same!

main reason:

different design aiming to allow unique_ptr to have same size as raw pointer making its deleter non-polymorphic

Default Deleter - unique_ptr

```
unique_ptr<A> uptr = make_unique<B>();
```

unique_ptr<A> can point to B -- BUT --
unique_ptr<A> cannot hold deleter of B...

```
void operator()(A* pa) {
    delete pa; // works OK if you have virtual dtor, which you should
}
```

Default Deleter - unique_ptr - how it works

```
template <class T, class D = default_delete<T>>
class unique_ptr_simplified: public D {
    unique_ptr(T*, D&); // providing a deleter, also as template parameter
    unique_ptr(T*);     // without providing a deleter, using default
};
```

Code: http://coliru.stacked-crooked.com/a/1a09853c5ec784e3

Default Deleter - unique_ptr

```
unique_ptr<A> ptr = make_unique<B>();
```

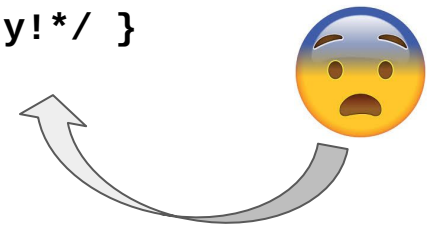
is actually:

```
unique_ptr<A, default_delete<A>> ptr =
    unique_ptr<B, default_delete<B>> {new B()};
```

calls:
```
template<typename Derived>
unique_ptr(unique_ptr<Derived>&& o) { *this = std::move(o); }
```

calls:
```
template<typename Derived>
default_delete(default_delete<Derived>&&) { /*empty!*/ }
```

BUT -- no harm if you have virtual destructor (which you should have)



Custom Deleter - unique_ptr

```
unique_ptr<A, DeleterA> ptr = unique_ptr<B, DeleterB>{new B(), deleterB};
```

Same problem, but now even more severe!!!

deleterB will not be called

Same code: http://coliru.stacked-crooked.com/a/1a09853c5ec784e3

Custom Deleter - unique_ptr

Could the language **disallow** slicing of custom deleter, so the following code would result with compilation error:

```
unique_ptr<A, DeleterA> ptr = unique_ptr<B, DeleterB>{new B(), deleterB};
```

But this would be OK:

```
    unique_ptr<A> ptr = make_unique<B>();
```

???

Custom Deleter - unique_ptr - disallow slicing

Yes, it could have been blocked:

```
template<typename T, typename OtherDeleter,
  std::enable_if_t<!std::is_same<OtherDeleter, Deleter>::value>* dummy =
nullptr>
auto& operator=(std::unique_ptr<T, OtherDeleter>&& other) = delete;
```

So why wasn’t it blocked?

See discussion here: https://stackoverflow.com/questions/56308336/why-unique-ptr-doesnt-prevent-slicing-of-custom-deleter

Almost done...

But what about shared_ptr?

Deleter - shared_ptr

```
shared_ptr<A> ptr = make_shared<B>();
```

How does it get to the proper destructor, even if not virtual?

Deleter - shared_ptr - type erasure

See: https://stackoverflow.com/questions/6324694/type-erasure-in-c-how-boostshared-ptr-and-boostfunction-work

```
struct deleter_base {
    virtual ~deleter_base() {}
    virtual void operator()( void* ) = 0;
};

template <typename T>
struct deleter : deleter_base {
    virtual void operator()( void* p ) {
        delete static_cast<T*>(p);
    }
};
```

(Very similar to the way std::any work)

Summary

shared_ptr: deleter is not part of the type, no slicing

unique_ptr: deleter is part of the type, beware of slicing

Resources and additional links

http://www.bourez.be/?p=19 or https://stackoverflow.com/questions/13460395/how-can-stdunique-ptr-have-no-size-overhead

https://www.bfilipek.com/2016/04/custom-deleters-for-c-smart-pointers.html & https://geidav.wordpress.com/tag/custom-deleter

https://stackoverflow.com/questions/28616141/what-is-the-rationale-for-the-difference-in-destruction-behavior-between-stdun

https://stackoverflow.com/questions/21355037/why-does-unique-ptr-take-two-template-parameters-when-shared-ptr-only-takes-one

https://www.geeksforgeeks.org/virtual-destruction-using-shared_ptr & https://stackoverflow.com/questions/3899790/shared-ptr-magic

https://stackoverflow.com/questions/36920908/c-shared-ptr-in-polymorphism-without-virtual-destructor

https://stackoverflow.com/questions/6634730/is-a-virtual-destructor-needed-for-your-interface-if-you-always-store-it-in-a-s

https://stackoverflow.com/questions/6324694/type-erasure-in-c-how-boostshared-ptr-and-boostfunction-work

https://stackoverflow.com/questions/56308336/why-unique-ptr-doesnt-prevent-slicing-of-custom-deleter

`+ Custom deleter simple usage example:` http://coliru.stacked-crooked.com/a/b4c0fdfae8c74d90

Thank you!

```
void conclude(auto greetings) {
    while(still_time() && have_questions()) {
        ask();
    }
    greetings();
}

conclude([]{ std::cout << "Thank you!"; });
```