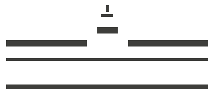


Applied
Mathematics
Münster

Smart Pointers in C++11

The very basics

living.knowledge
WWU Münster

Andreas Buhr

17. June 2015

std::shared_ptr‹T›

std::shared_ptr‹T›

- boost::shared_ptr‹T› since version 1.16 (2000)
- tr1::shared_ptr‹T› in standard (2007)
- std::shared_ptr‹T› in C++11 (2011)

Motivation

Automatic object lifetime management:

- ... no memory leaks
- ... no dangling pointers
- ... no double frees

Widely adopted: Some companies have “no new, no delete” coding policies.

Interface

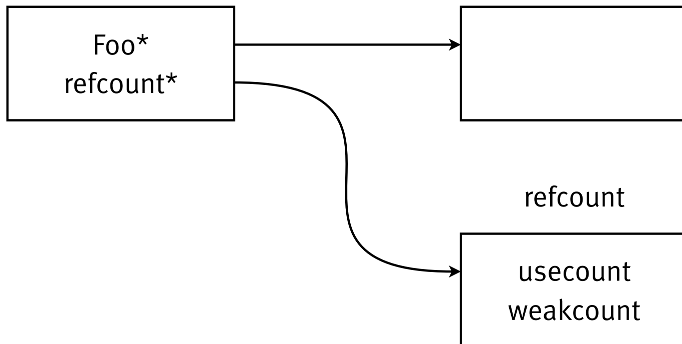
```
#include <memory>
class Foo { ... };

// creation:
std::shared_ptr<Foo> my_foo_ptr(new Foo(...));

// usage:
my_foo_ptr->some_function();
function_using_Foo(*my_foo_ptr);

// automatic delete
```

shared_ptr: How it Works



Things to know

- shared_ptr<T> is "as threadsafe as an int"
- Overhead for creation (allocate refcounter)
- Overhead for copy (update refcount, synchronization)
- Overhead for destruction

... there is no free lunch!

There is free lunch!

living.knowledge
WWU Münster

There is free lunch!

std::unique_ptr<T>

living.knowledge
WWU Münster

std::unique_ptr‹T›

```
#include <memory>
class Foo { ... };

// creation:
std::unique_ptr<Foo> my_foo_ptr(new Foo(...));

// usage:
my_foo_ptr->some_function();
function_using_Foo(*my_foo_ptr);

// automatic delete
```

living.knowledge
WWU Münster

std::unique_ptr‹T›

- Only one pointer points to an object

std::unique_ptr‹T›

- Only one pointer points to an object
- no copy constructor
- no assignment operator

living.knowledge WWU Münster

std::unique_ptr<T>

- Only one pointer points to an object
- no copy constructor
- no assignment operator
- but move constructor and move assignment operator

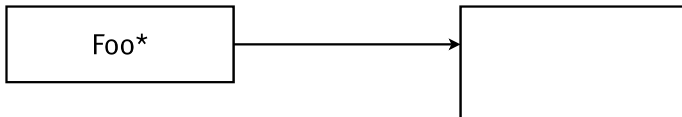
living.knowledge WWU Münster

std::unique_ptr‹T› move assignment

```
std::unique_ptr<int> a(new int(42));
std::unique_ptr<int> b;
// a now holds a pointer to 42; b is empty
b = std::move(a);
// now b holds the pointer to 42; a is empty
```

living.knowledge
WWU Münster

unique_ptr: How it Works



living.knowledge
WWU Münster

```
class Foo{ ... };
Foo* get_foo(void);
int treat_foo(Foo* theFoo);
```

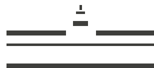
living.knowledge
WWU Münster

```
class Foo{ ... };
Foo* get_foo(void);
int treat_foo(Foo* theFoo);
int myfunction(void){
  std::unique_ptr<Foo> myfoo(get_foo());
  int result = treat_foo(myfoo.get());
  return result;
}
```

living.knowledge
WWU Münster

```
class Foo{ ... };
Foo* get_foo(void);
int treat_foo(Foo* theFoo);
int myfunction(void){
  std::unique_ptr<Foo> myfoo(get_foo());
  int result = treat_foo(myfoo.get());
  return result;
}
int myfunction(void){
  Foo* myfoo = get_foo();
  int result = treat_foo(myfoo);
  delete myfoo;
  return result;
}
```

```
class Foo{ ... };
Foo* get_foo(void);
int treat_foo(Foo* theFoo) noexcept;
int myfunction(void){
  std::unique_ptr<Foo> myfoo(get_foo());
  int result = treat_foo(myfoo.get());
  return result;
}
int myfunction(void){
  Foo* myfoo = get_foo();
  int result = treat_foo(myfoo);
  delete myfoo;
  return result;
}
```



```
class Foo{ ... };
Foo* get_foo(void);
int treat_foo(Foo* theFoo) noexcept;
int myfunction(void){
  std::unique_ptr<Foo> myfoo(get_foo());
  int result = treat_foo(myfoo.get());
  return result;
}
int myfunction(void){
  Foo* myfoo = get_foo();
  int result = treat_foo(myfoo);
  if(myfoo != nullptr)
    delete myfoo;
  return result;
}
```

living.knowledge
WWU Münster

```
class Foo{ ... };
Foo* get_foo(void);
int treat_foo(Foo* theFoo) noexcept;
int myfunction(void){
  std::unique_ptr<Foo> myfoo(get_foo());
  int result = treat_foo(myfoo.get());
  return result;
}
int myfunction(void){
  Foo* myfoo = get_foo();
  int result = treat_foo(myfoo);
  if(myfoo != nullptr)
    delete myfoo;
  return result;
}
```

Exactly the same binary code!

std::unique_ptr‹T› to std::shared_ptr‹T›

You can transfer ownership from a std::unique_ptr‹T› to a std::shared_ptr‹T›

living.knowledge
WWU Münster

std::unique_ptr‹T› to std::shared_ptr‹T›

You can transfer ownership from a std::unique_ptr‹T› to a std::shared_ptr‹T›

```
std::unique_ptr<int> a(new int(42));
std::shared_ptr<int> b;
b = std::move(a);
```

Advanced Topics (not today)

- make_shared
- make_unique
- custom deleter
- enable_shared_from_this
- weak_ptr

living.knowledge
WWU Münster

Take Home

1. Use smart pointers!
2. Make std::unique_ptr your default.