

C++ 11 Smart pointers

Administrative

- Next project soon
- Read the online stuff for this week, especially material on smart pointers

Outline

- 4 kinds of smart pointers
- Benefits of Smart pointers
- Examples
- Move verses copy
- Git projects

4 kinds of smart pointers

- `std::auto_ptr` ← Deprecated, does not work correctly

- `std::unique_ptr` ← Can have only 1 unique pointer to an object (but can transfer ownership From `unique_ptr` to `unique_ptr`)

- `std::shared_ptr` ← Can have multiple shared pointers to same object

- `std::weak_ptr` ← Not discussed in this class see 'Modern effective C++' By Scott Meyers for full explanation

Benefits

- Do not need to worry about deleting
- Or dangling pointers
- Or copies (deep, shallow or otherwise)
- Work very well with Standard Library

Example – the risky way

//A.h

```
class A {  
    public:  
        A();  
        ~A();  
    private:  
        int* i;  
        // disable value-copying  
        A(const A&);  
        A& operator=(const A&);  
};
```

//A.cpp

```
A::A():i(new int(0)) { }  
  
A::~~A() {  
    if(i)  
        delete i;  
}
```

Quite a bit of boiler plate code

Example – the new and improved way

//A.h

```
class A {  
    public:  
        A();  
    private:  
        unique_ptr<int> i;  
};
```

//A.cpp

```
A::A():i(new int(0)) { }
```

Look at that code savings!

Default destructor is fine so do not have to define it.

Copying automatically disabled (unique_ptr is not copyable)

Move defined automatically (unique_ptr is moveable)

Whats move?

Move in C++ 11

- Pre C++ 11 has 4 functions that you must manage if your object holds dynamic data
 - Constructor, destructor, copy constructor, assignment operator
- C++ 11 adds 2 more
 - Move constructor
 - Move assignment
- These let you move objects instead of copying. **A big savings if object is large**

Move constructor and move assignment

- Do I have to define these?
- Not for this class!!
- The syntax is complex, the gains are substantial but the reasons are subtle.
- See 'Move Semantics' on course website

Example git projects

- demo_pointers_vectors
 - unique_ptr in a vector
 - No need to delete
- unique_ptr_ex
 - Difference between class A and B
 - Using unique_ptr instead of raw ptr

Summary

- Try to use `unique_ptr` for your pointer needs
- Much safer
 - Don't have to worry about leaking memory
 - Or dangling pointers
 - Or getting boilerplate code correct (no copy constructor or assignment operator)
 - Or shallow or deep copies