C++ smart pointers

- Handled through the templated shared_ptr class in the tr1/memory library
- Internally track reference counts and pointer to actual item
- shared_ptr copy constructor and assignment operator adjust reference counts appropriately for both smart pointers (source and destination)
- Automatically deallocate resource when reference count 0

Syntax

- Include smart pointer library

```
#include <tr1/memory>
```

- Declare smart pointer for desired resource, e.g. for a List

```
std::tr1::shared_ptr<List> ptrA;
```

- Request new instance of resource

```
ptrA.reset(new List);
```

Syntax (cont.)

- Access resource fields/methods through smart pointer

```
ptrA->insert(10);
```

- Copy between smart pointers (updates both ref counts), implicitly happens when pass smart pointers as params

```
ptrB = ptrA;
```

- “nullify” a smart pointer (make it point nowhere)

```
ptrA.reset();
```

Weak pointers

- A templated weak_ptr also supported, that allows controlled access to item but does not change reference count
- Lock method in weak_ptr only allows access if safe (item still exists)

```
Std::tr1::weak_ptr<List> wptr = ptrA; // make wk ptr refer to existing item
....
If (std::tr1::shared_ptr<List> tmp = wptr.lock()) { // try to access thru weak ptr
    tmp->print(); // only gets access to this list method if lock ok'd it
}
```