RAII & Smart Pointers

CS 247

University of Waterloo

cs247@uwaterloo.ca

June 4, 2018
1 RAI
   - Three levels of Guarantee
   - What’s RAI

2 Smart Pointers
   - Basic Information
   - How to use them
Three levels of Guarantee

- Basic guarantee: if an exception is thrown, data will be in a valid but unspecified state.
- Strong guarantee: if an exception is thrown, no side-effects will have occurred, your program's state is as if the function was never called.
- No-throw guarantee: an exception is never thrown.
RAII: Resource Acquisition is Initialization
RAII is vital to writing exception-safe code in C++.

Mechanism: When acquiring a resource you should also be initializing a statically allocated object with it.

When the objects lifetime is over its destructor should free the resource.

e.g. files (fstreams), memory (smart pointers), and more.
Smart Pointers

- `unique_ptr` is for sole ownership, only one `unique_ptr` can point to the same block of heap memory.
- `shared_ptr` is for shared ownership, allows many pointers that all point to the same block of heap memory and only deletes that memory when no other `shared_ptrs` point to it.
- `weak_ptr` is for non-owning referencing, created with an existing `shared_ptr`, used mainly to break cyclical references between `shared_ptrs`.

Example: `smartptr.cc`
- In most cases use smart pointers as you would raw pointers.

class MyClass {
    unique_ptr<Impl> pImpl;
    ...;
};
**Shared Pointers**

- `shared_ptr` works by maintaining a reference count to an object.
- For each `shared_ptr` you create to the same object the reference count is incremented for that piece of data.
- When the destructor of a `shared_ptr` executes it decrements the reference count, if at that point the reference count is 0 then it frees the memory.
- This only works if the `shared_ptr` class is used properly.
- Let's consider how we might implement `shared_ptr` for ints.
The End