Homework 16: Move Semantics, Value Categories, and Smart Pointers

Instructor: Mehmet Emre

CS 32 Spring '22

Due: 6/1 12:30pm
Name & Perm #: Bharat Kathi (5938444)

Homework buddy (leave blank if you worked alone):
Reading: The relevant resources mentioned in lecture 14.

1

In class, we explored what `std::move` does, and did some experiments to show that `std::move(x)` doesn't actually move `x` anywhere. What is the semantics of `std::move` (that is, what does `std::move` actually do)?

Answer: std::move changes the expression from being a lvalue to a rvalue by returning the rvalue reference to the given argument. The compiler will chose to use the move assignment operator and move constructors over and copy assignment and copy constructor to actually move the values.

2 2 pts

In the `UniquePtr` class we developed in class, how we changed the copy and move constructors as well as the assignment operators to guarantee that only 1 `UniquePtr` can point to an object at a time.

How do the move constructor and the move assignment operator provide this guarantee?

Answer: The move copy constructor sets the other object to null. Then the move assignment operator deletes the previous object before making the new assignment.

3

In the class, I argued that `std::unique_ptr` is a good first choice when needing a pointer.

1. (1 pt) What is the advantage of `std::unique_ptr` over a plain pointer (like `T *`)?

   Answer: std::uniqueptr has all the benefits of an optional. This means that it takes care of memory management while also acting as a pointer. It also has a move contstructor and move assignment operator instead of the copy versions.

2. (2 pt) What are the two advantages of using `std::unique_ptr` over `std::shared_ptr`?

   Answer: One advantage is that unique ptr will not leak memory when creating cyclic references. Another advantage is that unique ptr is not affected by the resource intensive increment and decrement reference costs, since it doesn't need reference counting.

4

Reference counting cannot detect that certain objects are not reachable (and can be safely destroyed).

1. (2 pt) What causes this limitation (how the objects should be used for reference counting to fail to determined that they are unreachable)?

   Answer: This is caused when objects are used to create cyclic references, which is using pointers between objects which point to each other.

2. (1 pt) What machinery is provided by the C++ standard library to get around this limitation?

   Answer: `weak_ptr`