Smart Pointers

TOPICS

- std::shared_ptr
- std::unique_ptr
- std::weak_ptr

TOPICS

- std::shared_ptr
- std::unique_ptr
- std::weak_ptr

SMART PONTERS

- Smart pointers are objects that behave like pointers, but with added features such as automatic memory management, increased code safety, and flexibility.
- Smart pointers come in a variety of types, each with its own unique features and benefits.
  - The shared_ptr type supports shared ownership.
    - Counts the number of owners.
    - When count is zero (all owners have released ownership), the object is deleted.
  - The unique_ptr maintains a unique instance of an object via a pointer.
    - No reference counting.
    - When moved, the original pointer is set to null.
    - Copy not allowed
  - The weak_ptr type refer to a weak reference to memory.
    - Weak pointers create a shared reference without adding to the reference count.
    - Create a weak pointer to optionally preserve a pointer in memory

STD::UNIQUE_PTR

- When you need a smart pointer for a plain C++ object, use unique_ptr
- Unlike share_ptr, a unique_ptr does not share its pointer.

Syntax:

```
    unique_ptr<double> sp1(new double(100));
            or
    unique_ptr<double> sp2 = std::make_unique<int>(5);  /* introduced in C++ 14 */
```

STD::SHARED_PTR

- Use a shared_ptr when more than one owner might have to manage the lifetime of the object in memory.
- After you initialize a shared_ptr you can copy it, pass it by value in function arguments, and assign it to other shared_ptr instances

Syntax:

```
shared_ptr<double> sp1(new double(100));
        or
shared_ptr<double> sp2 = std::make_shared<int>(5);
```

STD::WEAK_PTR

Provides a way to access the underlying object of a shared_ptr without causing the reference count to be incremented.

- Typically, this need arises when you have cyclic references between shared_ptr instances.
- By using a weak_ptr, you can create a shared_ptr that joins to an existing set of related instances, but only if the underlying memory resource is still valid.

Syntax:

```
shared_ptr<double> sp(new double(100));
weak_ptr<double> wp(sp);
```

ADDITIONAL FEATURES

- Both unique_ptr and shared_ptr allow you to specify a custom deleter function or function object that will be called when the pointer is deleted.
- shared_ptr allows you to specify a custom allocator object that will be used to allocate memory for the reference count and control block associated with the pointer.
- With a shared_ptr you can specify a custom hash function.
  - This is useful as a key in an unordered container like unordered_map.
- If you're comparing smart pointers with == or != operators, you can specify a custom comparison operator that will be used to compare the underlying raw pointers.