Shared Pointer Modern C++

By

SimplyProgram.Hub

Ashish Shelke

Smart Pointer

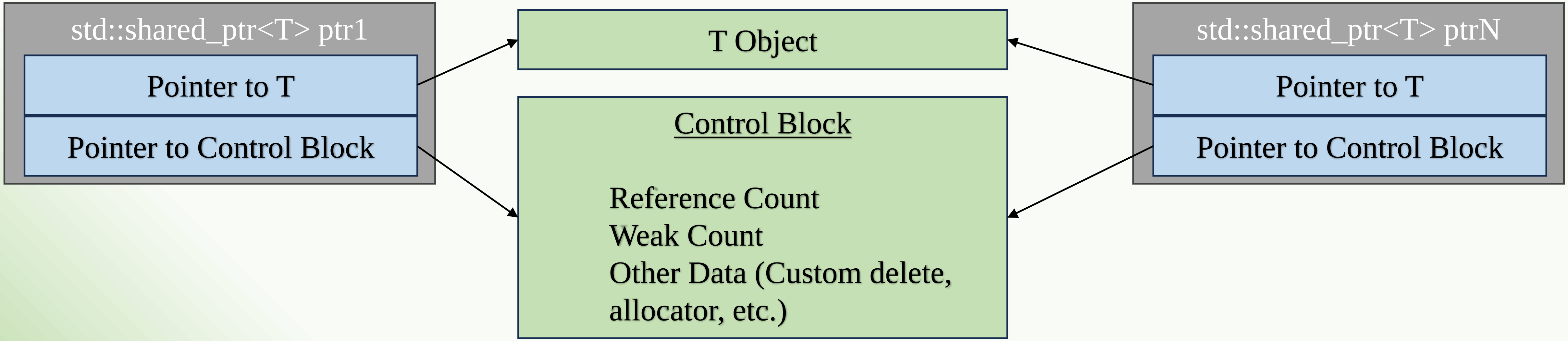
- Smart pointer is wrapper around the raw pointer that acts like raw pointer but avoids memory leaks and segmentation faults.
- Smart pointers used to make sure that an object (dynamically allocated) is deleted. Such objects are destroyed in appropriate manner and at appropriate time.

- There are four smart pointers (#include <memory>)
  1. std::auto_ptr
  2. std::unique_ptr
  3. std::shared_ptr
  4. std::weak_ptr

Shared Pointer

- An object accessed by std::shared_ptr has its lifetime managed by shared ownership with other std::shared_ptr
- No specific std::shared_ptr owns object ownership.
- All std::shared_ptr pointing to object, collaborate to ensure object’s destruction when object is no longer needed.
- When the last shared_ptr pointing to an object stops pointing there, that shared_ptr destroys the object it points to.
- std::shared_ptr has resource’s reference count, this keeps track of how many std::shared_ptr are pointing to respective resource.
- This reference count, deleter functions are stored in ‘control block’. Address to control block is stored in std::shared_ptr<T> object.

Shared Pointer's Object's Memory



- While creating 1st std::shared_ptr for resource, memory for control block is dynamically allocated.
- 2nd onwards created std::shared_ptr, points to resource and this control block

Rules for creating control block

- std::make_shared always creates a control block.
    - It allocates a single chunk of memory to hold both object and control block
    - It helps to optimize code and increase speed of the executable

- When std::shared_ptr is constructed from a unique-ownership pointer.
    - During construction of std::shared_ptr, std::unique_ptr is assigned with null.

- When std::shared_ptr constructor is called from a raw pointer, it creates a control block.
    - It may lead to undefined behavior.
    - 2nd onward std::shared_ptr created with raw pointer, creates newer control block.
    - So same resource is pointed with multiple control block
    - So raw pointer assignment should be avoided.

Deleter

- Default resource destruction is via delete operator

- Unlike std::unique_ptr custom delete function is provided in construction call of std::shared_ptr

  std::shared_ptr<T> ptr (object, deleter_function)

- Each std::shared_ptr which is pointing to same resource can have different custom delete function

std::weak_ptr

- std::weak_ptr are created from std::shared_ptr.
- It points to same object as std::shared_ptr is initialized with, std::weak_ptr don’t affect reference count of the object.
- It holds a non-owning (‘weak’) reference to an object that is managed by std::shared_ptr
- If std::shared_ptr is pointing to some resource and this resource might be deleted by someone else, std::weak_ptr is used to track the object. This std::weak_ptr is converted to std::shared_ptr to acquire ownership with *lock()* member function.
- std::weak_ptr can’t be dereferenced, nor can be tested for nullness.

```
std::shared_ptr<T> spw =  std::make_shared<T> ();
std::weak_ptr<T> wpw (spw);
spw = nullptr;
if(wpw.expired())     // Should be checked before created std::shared_ptr else exception is thrown
        std::shared_ptr<T> spw1 = wpw.lock();
```

std::make_shared()

- It constructs an object of type T and wraps it in a std::shared_ptr

```
template< class T, class... Args >
shared_ptr<T> make_shared( Args&&... args );
```

- Return type is std::shared_ptr<T>
- std::make_shared is inappropriate when need to specify custom delete function and desired to pass braced initializers

- Examples:

```
std:: shared _ptr<T> ptr = std::make_shared<T>();
```