

# Shared Pointer Solutions

# std::shared\_ptr

- Briefly describe std::shared\_ptr
  - std::shared\_ptr is a "smart pointer" class
  - Unlike std::unique\_ptr, a shared\_ptr object can be copied and assigned to
  - This allows multiple shared\_ptr objects to have access to the allocated memory
  - When the last shared\_ptr object that accesses the shared memory is destroyed, the memory is released

# std::shared\_ptr structure

- Briefly describe the structure of the shared\_ptr class
  - shared\_ptr has a member which is a pointer to the shared memory
  - It has another member which is a pointer to a "control block"
  - The control block contains a reference counter which keeps track of the number of objects that can access the shared memory
  - Every time one of these objects is copied or assigned from, the reference counter is incremented
  - Every time one of these objects is destroyed, the reference counter is decremented
  - When the last of the objects is destroyed, the counter goes to zero and the shared memory is released

# Initialization

- Give two ways to create an initialized `std::shared_ptr` object
  - Call `make_shared()`
  - Pass a pointer as argument to the `shared_ptr` constructor
- Is there any reason to prefer one approach over the other?
  - `make_shared()` will make a single call to `new()` to allocate the shared memory and the control block in a single, contiguous location in memory
  - The constructor call will make a second call to `new()` to allocate the control block
  - This will probably have a different address from the first pointer, and the processor will not be able to optimize the data fetches into a single operation
  - `make_shared()` should be preferred as it avoids this extra overhead

# shared\_ptr copying

- Describe what happens when a shared\_ptr object is created as a copy of another object
  - The new object will share its pointer member and control block with those in the original object
  - The reference counter in the shared control block will be incremented

# shared\_ptr assignment

- Describe what happens when a shared\_ptr object is assigned to another shared\_ptr object
  - If the two objects have shared the same shared memory, the reference counter will be incremented
  - If they refer to different shared memory, the assigned-to object will decrement its reference counter
  - If this causes its reference counter to become zero, its shared memory will be released
  - A copy is then performed - the assigned-to object shares the allocated memory and control block, and the reference counter in the shared control block is incremented

# shared\_ptr operations

- Write a simple program which creates and initializes shared\_ptr object and performs some operations on it

# shared\_ptr and threads

- What issues arise when `shared_ptr` is used in a multithreaded program?
  - The reference counter is atomic
  - This prevents data races when a `shared_ptr` object is copied or assigned to
  - However, the data in the allocated memory requires protection if there are conflicting accesses to it
- What impact do these issues have on the performance of `shared_ptr`?
  - Using atomic operations on the reference counter adds a significant amount of overhead to `shared_ptr`
  - Normally, `unique_ptr` should be used instead, unless the extra features of `shared_ptr` are needed



# Shared Pointer Applications

- Give an example of a programming problem where `shared_ptr` could be useful
  - In applications which have many copies of the same data in allocated memory, a `shared_ptr` will save memory by only creating a single copy of the data and then using multiple references to this single copy
  - Duplicate words in large documents: we can save memory by storing them in a `shared_ptr`
  - A web browser where several tabs are displaying the same logo: we can save memory by storing the image in a `shared_ptr`