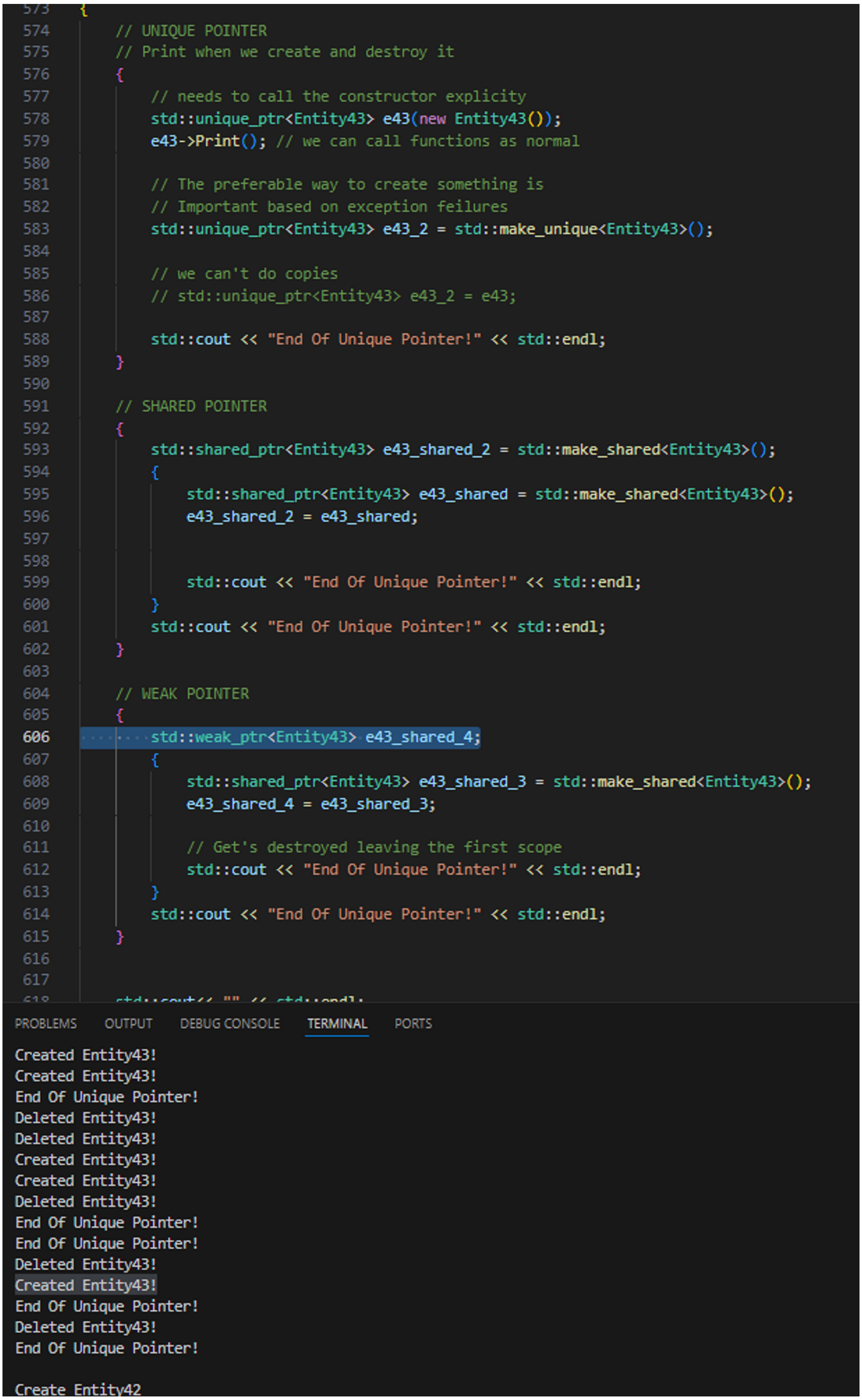
43 - SMART POINTERS in C++ (std::unique_ptr, std::shared_ptr, std::weak_ptr)

segunda-feira, 10 de março de 2025 07:05

- smart pointers is a ay to automate the process of free memory when allocated with the new statement
  - when you call new, you don't need to call delete
- Wrapper of a new pointer, so you don't worry to free it
- UNIQUE_POINTER
  - scoped pointer, when that pointer goes out of scope it'll get destroed and deleted
  - Only leave in the scope
  - must be unique, can't copy it… otherwise 2 pointers will poont to the same memory. And when one die, it'l free that memory and the other pointer has no value anymore
  - Only referen to that pointer
  - sTACK ALLOCATED OBJECT
  - the problem is to copy or pass it to a function because we can't opy it
- SHARED POINTERS
  - up to the compiler how it works
  - Works by reference cointers, keep tracking of how many pointers points to that emory
  - When it get's to zero is when that memory is free
    - Create 1 shared pointer, copy it
      - 2 reference
    - when both of them are free, the memory is free
  - when al of the references get out of scope, the memory allocation dies
  - wHEN ASSIGN A REF COINTER TO ANOTHER SHARED POINT you are copying it and increasing count
- WEAK POINTER
  - When assign a shared pointer to a week pointer, it doesn't increase counter
  - great when you don't want to take ownershi of the entity,
  - just store a reference to that
  - is it even alive, if yes you do what you need to do. but it won't keep t alive
- Try to use them all the time. They are usefull
- Auto control of the memory menamegent



```cpp
{
    // UNIQUE POINTER
    // Print when we create and destroy it
    {
        // needs to call the constructor explicity
        std::unique_ptr<Entity43> e43(new Entity43());
        e43->Print(); // we can call functions as normal

        // The preferable way to create something is
        // Important based on exception feilures
        std::unique_ptr<Entity43> e43_2 = std::make_unique<Entity43>();

        // we can't do copies
        // std::unique_ptr<Entity43> e43_2 = e43;

        std::cout << "End Of Unique Pointer!" << std::endl;
    }

    // SHARED POINTER
    {
        std::shared_ptr<Entity43> e43_shared_2 = std::make_shared<Entity43>();
        {
            std::shared_ptr<Entity43> e43_shared = std::make_shared<Entity43>();
            e43_shared_2 = e43_shared;


            std::cout << "End Of Unique Pointer!" << std::endl;
        }
        std::cout << "End Of Unique Pointer!" << std::endl;
    }

    // WEAK POINTER
    {
        std::weak_ptr<Entity43> e43_shared_4;
        {
            std::shared_ptr<Entity43> e43_shared_3 = std::make_shared<Entity43>();
            e43_shared_4 = e43_shared_3;

            // Get's destroyed leaving the first scope
            std::cout << "End Of Unique Pointer!" << std::endl;
        }
        std::cout << "End Of Unique Pointer!" << std::endl;
    }


    std::cout<< "" << std::endl;
```

```
Created Entity43!
Created Entity43!
End Of Unique Pointer!
Deleted Entity43!
Deleted Entity43!
Created Entity43!
Created Entity43!
Deleted Entity43!
End Of Unique Pointer!
End Of Unique Pointer!
Deleted Entity43!
Created Entity43!
End Of Unique Pointer!
Deleted Entity43!
End Of Unique Pointer!

Create Entity42
```