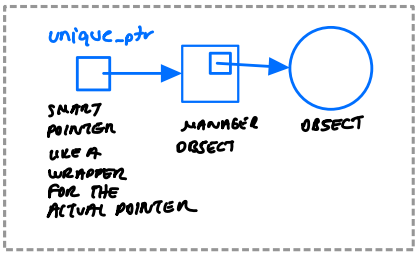
SMART POINTERS

- ONE BIG CHANGE THAT COMES WITH C++11 IS THAT WE DON'T NEED TO MANUALLY DELETE.
- THANKS TO : shared_ptr, unique_ptr and weak_ptr
- TO USE THESE CLASSES, INCLUDE <memory>

unique_ptr

SIMPLY HOLDS A POINTER, AND ENSURES THAT THE POINTER IS DELETED ON DESTRUCTION.

unique_ptr OBJECTS CANNOT BE COPIED

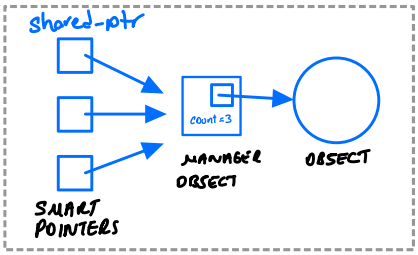


shared_ptr

FUNCTIONS THE SAME WAY AS unique_ptr

UNLIKE unique_ptr, IT ALLOWS COPYING OF THE shared_ptr OBJECT TO ANOTHER shared_ptr, AND THEN ENSURES THAT THE POINTER IS STILL GUARANTEED TO ALWAYS BE DELETED ONCE ALL shared_ptr OBJECTS THAT WERE HOLDING IT ARE DESTROYED.

IT KEEPS A SHARED COUNT OF HOW MANY shared_ptr OBJECTS ARE HOLDING THE SAME POINTER



```cpp
#include <iostream>
#include <memory>

using namespace std;

class IntCell{
    private:
        int* cell;
    public:
        IntCell(int x){
            cout<<"In constructor"<<endl;
            cell = new int(x);
        }

        ~IntCell(){
            cout<<"In destructor"<<endl;
            delete cell;
        }
        int read(){
            return *cell;
        }

};
```

```cpp
int main(){

    //--> unique_ptr
    IntCell* p1 = new IntCell(3);
    delete p1;

    std::unique_ptr<IntCell> p2 = make_unique<IntCell>(7);
    //deletes the object automatically when the method returns

    std::unique_ptr<IntCell> p3(new IntCell(8));

    //p2 = p3; //invalid
    p2 = move(p3);
    cout<<p2->read()<<endl;
    //cout<<p3->read()<<endl; //invalid access

    //-->shared_ptr
    std::shared_ptr<IntCell> s1 = make_shared<IntCell>(9);
    std::shared_ptr<IntCell> s2 = make_shared<IntCell>(7);
    s1 = s2;
    cout<<s1->read()<<endl;
    cout<<s2->read()<<endl;
    cout<<"count:"<<s1.use_count()<<endl;
    std::shared_ptr<IntCell> s3 = s1;
    cout<<"count:"<<s1.use_count()<<endl;

    return 0;
}
```

⇒

```
In constructor
In destructor
In constructor
In constructor
In destructor
8
In constructor
In constructor
In destructor
7
7
count:2
count:3
In destructor
In destructor
```