

## - Example

This example demonstrates cyclic references when using `std::shared_ptr`.

### WE'LL COVER THE FOLLOWING ^

- Example
- Explanation

## Example #

```
// cyclicReference.cpp

#include <iostream>
#include <memory>

struct Son;
struct Daughter;

struct Mother{
    ~Mother(){
        std::cout << "Mother gone" << std::endl;
    }
    void setSon(const std::shared_ptr<Son> s ){
        mySon=s;
    }
    void setDaughter(const std::shared_ptr<Daughter> d ){
        myDaughter=d;
    }
    std::shared_ptr<const Son> mySon;
    std::weak_ptr<const Daughter> myDaughter;
};

struct Son{
    Son(std::shared_ptr<Mother> m):myMother(m){}
    ~Son(){
        std::cout << "Son gone" << std::endl;
    }
    std::shared_ptr<const Mother> myMother;
};

struct Daughter{
    Daughter(std::shared_ptr<Mother> m):myMother(m){}
    ~Daughter(){
        std::cout << "Daughter gone" << std::endl;
    }
}
```

```

    std::shared_ptr<const Mother> myMother;
};

int main(){
    std::cout << std::endl;
    {
        std::shared_ptr<Mother> mother= std::shared_ptr<Mother>( new Mother);
        std::shared_ptr<Son> son= std::shared_ptr<Son>( new Son(mother) );
        std::shared_ptr<Daughter> daughter= std::shared_ptr<Daughter>( new Daughter(mother) );
        mother->setSon(son);
        mother->setDaughter(daughter);
    }
    std::cout << std::endl;
}

```



## Explanation #

- In line 41 – 47, due to the artificial scope, the lifetime of the mother, the son, and the daughter are limited. In other words, `mother`, `son`, and `daughter` go out of scope, and therefore the destructor of the class `Mother` (line 10 - 12), `Son` (line 25 - 27), and `Daughter` (line 33 - 35) should automatically be invoked.
- We state *should*, because only the destructor of the class `Daughter` is called.
- The graphic of the source code shows that we have a cyclic reference of `std::shared_ptr` between mother and son. Therefore, the reference counter is always greater than 0, and the destructor will not automatically be invoked.
- That observation does not hold true for mother and daughter. If the daughter goes out of scope, the reference counter of the `std::shared_ptr myMother` (line 36) becomes 0 and the resource will automatically be deleted.

---

Test your understanding with an exercise in the next lesson.