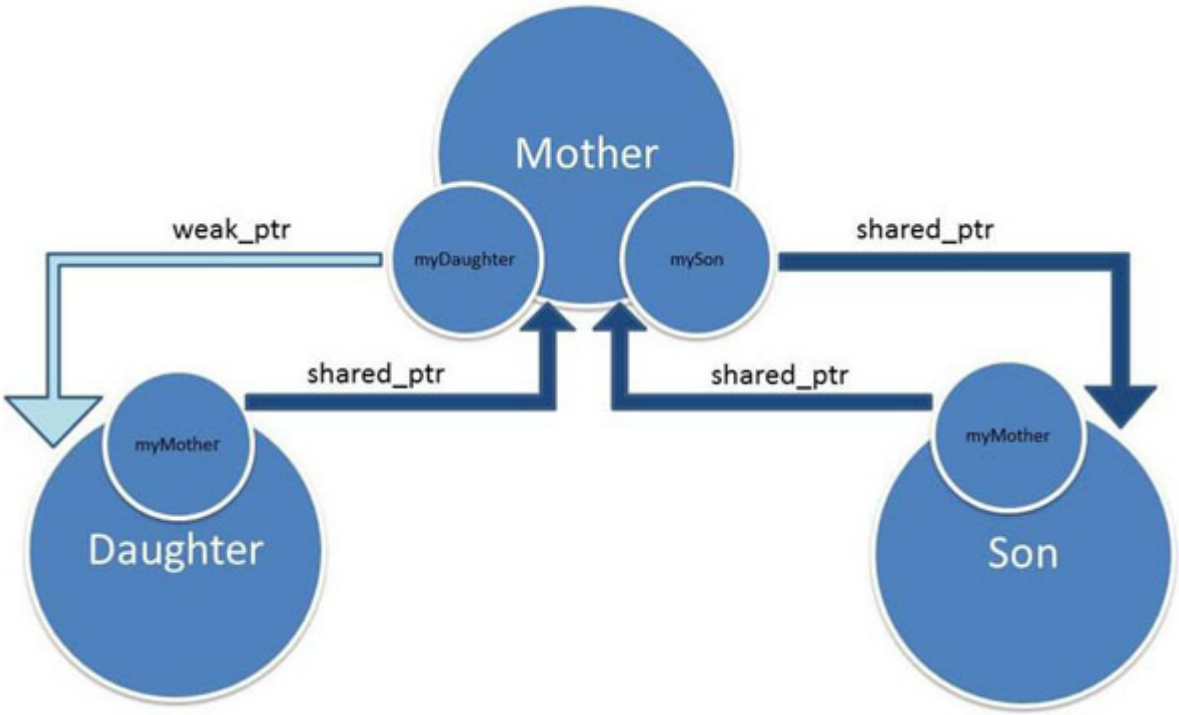
# Cyclic References

In this lesson, we'll see how the use of shared pointers can create a reference cycle and why this could be harmful.

You get cyclic references of `std::shared_ptr` if they refer to each other. So, the resource counter never becomes 0, and the resource is not automatically released. You can break this cycle if you embed an `std::weak_ptr` in the cycle. `std::weak_ptr` does not modify the reference counter.

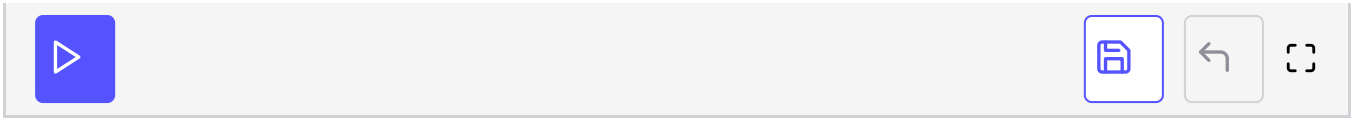
The result of the code sample is that the daughter is automatically released, but neither the son nor the mother is. The mother refers to her son via an `std::shared_ptr` and to her daughter via an `std::weak_ptr`. Maybe it helps to see the structure of the code in an image.




Finally the source code:

```
// cyclicReference.cpp
#include <iostream>
#include <memory>

using namespace std;
```

```
struct Son;
struct Daughter;


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

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

struct Daughter{
Daughter(shared_ptr<Mother> m):myMother(m){}
  ~Daughter(){cout << "Daughter gone" << endl;}
  shared_ptr<const Mother> myMother;
};
int main()
{
  shared_ptr<Mother> mother= shared_ptr<Mother>(new Mother);
  shared_ptr<Son> son= shared_ptr<Son>(new Son(mother) );
  shared_ptr<Daughter> daugh= shared_ptr<Daughter>(new Daughter(mother));
  mother->setSon(son);
  mother->setDaughter(daugh);
  return 0;
}                                // Daughter gone
```



Cyclic references

In the next lesson, we will learn about type traits in the C++ Standard Library.