

# Weak Pointers

This is the last component of the smart pointer family. Its purpose is much more limited compared to the others.

To be honest, `std::weak_ptr` is not a smart pointer. `std::weak_ptr` supports no transparent access to the resource because it only borrows the resource from a `std::shared_ptr`. `std::weak_ptr` does not change the reference counter:

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

int main(){
    auto sharedPtr= std::make_shared<int>(2011);
    std::weak_ptr<int> weakPtr(sharedPtr);

    std::cout << weakPtr.use_count() << std::endl;    // 1
    std::cout << sharedPtr.use_count() << std::endl;    // 1

    std::cout << weakPtr.expired() << std::endl;        // false
    if( std::shared_ptr<int> sharedPtr1= weakPtr.lock() ) {
        std::cout << *sharedPtr << std::endl; // 2011
    }
    else{
        std::cout << "Don't get it!" << std::endl;
    }

    weakPtr.reset();

    if( std::shared_ptr<int> sharedPtr1= weakPtr.lock() ) {
        std::cout << *sharedPtr << std::endl;
    }
    else{
        std::cout << "Don't get it!" << std::endl;    // Don't get it!
    }

    return 0;
}
```



The table provides an overview of the methods of `std::weak_ptr`.

Name	Description
------	-------------

<code>expired</code>	Checks if the resource was deleted.
<code>lock</code>	Creates a <code>std::shared_ptr</code> on the resource.
<code>reset</code>	Resets the resource
<code>swap</code>	Swaps the resources.
<code>use_count</code>	Returns the value of the reference counter.

### Methods of `std::weak_ptr`

There is one reason for the existence of `std::weak_ptr`. It breaks the cycle of `std::shared_ptr`. We will discuss these cyclic references in detail in the next lesson.