- Example

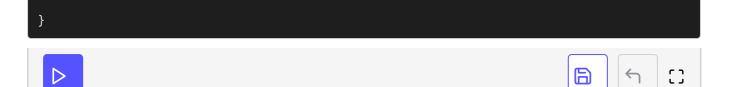
As compared to the std::shared_ptr, std::weak_ptr does not change the reference counter of the shared variable. Let's take a look at that feature in the example below.

WE'LL COVER THE FOLLOWING ^

- Example
 - Explanation

Example

```
// weakPtr.cpp
#include <iostream>
#include <memory>
int main(){
  std::cout << std::boolalpha << std::endl;</pre>
  auto sharedPtr=std::make_shared<int>(2011);
  std::weak ptr<int> weakPtr(sharedPtr);
  std::cout << "weakPtr.use_count(): " << weakPtr.use_count() << std::endl;</pre>
  std::cout << "sharedPtr.use_count(): " << sharedPtr.use_count() << std::endl;</pre>
  std::cout << "weakPtr.expired(): " << weakPtr.expired() << std::endl;</pre>
  if( std::shared_ptr<int> sharedPtr1 = weakPtr.lock() ) {
    std::cout << "*sharedPtr: " << *sharedPtr << std::endl;</pre>
    std::cout << "sharedPtr1.use_count(): " << sharedPtr1.use_count() << std::endl;</pre>
  else{
    std::cout << "Don't get the resource!" << std::endl;</pre>
  weakPtr.reset();
  if( std::shared_ptr<int> sharedPtr1 = weakPtr.lock() ) {
    std::cout << "*sharedPtr: " << *sharedPtr << std::endl;</pre>
    std::cout << "sharedPtr1.use_count(): " << sharedPtr1.use_count() << std::endl;</pre>
  else{
    std::cout << "Don't get the resource!" << std::endl;</pre>
```



Explanation

- In line 11, we create a std::weak_ptr that borrows the resource from the std::shared ptr.
- The output of the program shows that the reference counter is 1 (line 13 and 14), meaning that std::weak does not increment the counter.
- The call weakPtr.expired() checks if the resource was already deleted.

 That is equivalent to the expression weakPtr.use_count() == 0.
- If the std::weak_ptr shared a resource, you can use weakPtr.lock() to create a std::shared_ptr out of it.
- The reference counter will now be increased to 2 (line 18). After resetting the weakPtr (line 25), the call weakPtr.lock() fails.

That was almost the whole story for the std::weak_ptr. Almost, because the
std::weak_ptr has a special job: it helps to break the cyclic references of
std::shared_ptr.

In the next lesson, we will take a look at the issue of cyclic references while using std::shared_ptr.