- Solution

Let's have a look at the solution to the previous exercise in this lesson.

WE'LL COVER THE FOLLOWING

Solution Review

```
// templateTypeTraitsModifications.cpp

#include <iostream>
#include <type_traits>

int main(){

  std::cout << std::boolalpha << std::endl;

  std::cout << "std::is_const<std::add_const<int>::type>::value: " <<  std::is_const<std::add
  std::cout << "std::is_const<std::remove_const<const int>::type>::value: " << std::is_const<

  std::cout << std::endl;
  typedef std::add_const<int>::type myConstInt;
  std::cout << "std::is_const<myConstInt>::value: " << std::is_const<myConstInt>::value << st
  typedef const int myConstInt2;
  std::cout << "std::is_same<myConstInt, myConstInt2>::value: " << std::is_same<myConstInt, m

  std::cout << std::endl;

  int fir= 1;
  int& refFir1= fir;
  using refToIntType= typename std::add_lvalue_reference<int>::type;
  refToIntType refFir2= fir;

  std::cout << "(fir, refFi1r, refFir2): "  << "(" << fir << ", " << refFir1 << ", " << refFi

  fir= 2;

  std::cout << "(fir, refFir1, refFir2): "  << "(" << fir << ", " << refFir1 << ", " << refFi

  std::cout << std::endl;

}
```

Explanation

Lines 10 and 11 determine at compile-time if the manipulated type is `const`. `myConstInt` is a type alias for an `const int`. Line 17 shows, with the help of the function `std::is_same`, that `myConstInt` and `myConstInt2` are the same types. An lvalue such as `fir` can only be bound to an `lvalue reference`. Line 24 shows this in a complicated way. Line 26 proves that `refFir1` and `refFire2` are references.

In the next lesson, we'll discuss constexpr.