- Exercise

In this lesson, we'll solve an exercise related to the automatic return type.

WE'LL COVER THE FOLLOWING

Problem Statement

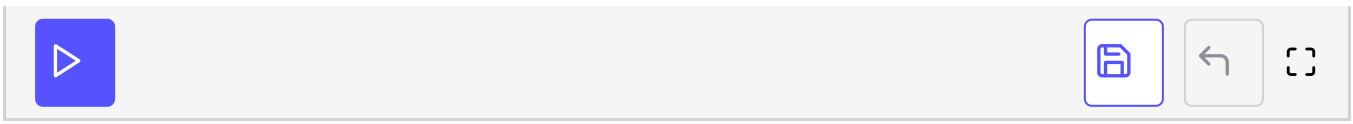
The two algorithms `gcdConditional` and `gcdCommon`, in the given code, use the type-traits library.

- Study both algorithms.
- Both algorithms determine their return type at compile-time.

Why is it not possible to use `auto` or `decltype(auto)` to automatically deduce the return type?

```
#include <iostream>
#include <string>
#include <type_traits>
#include <typeinfo>

template<typename T1, typename T2,
         typename R = typename std::conditional <(sizeof(T1) < sizeof(T2)), T1, T2>::type>
R gcdConditional(T1 a, T2 b){
  static_assert(std::is_integral<T1>::value, "T1 should be an integral type!");
  static_assert(std::is_integral<T2>::value, "T2 should be an integral type!");
  if( b == 0 ){ return a; }
  else{
    return gcdConditional(b, a % b);
  }
}


template<typename T1, typename T2,
         typename R = typename std::common_type<T1, T2>::type>
R gcdCommon(T1 a, T2 b){
  static_assert(std::is_integral<T1>::value, "T1 should be an integral type!");
  static_assert(std::is_integral<T2>::value, "T2 should be an integral type!");
  if( b == 0 ){ return a; }
  else{
```

```
    return gcdCommon(b, a % b);
  }
}

int main(){


  std::cout << std::endl;

  std::cout << "gcdConditional(100, 10LL) = " << gcdConditional(100, 10LL) << std::endl;
  std::cout << "gcdCommon(100, 10LL) = " << gcdCommon(100, 10LL) << std::endl;

  std::conditional <(sizeof(int) < sizeof(long long)), int, long long>::type gcd1 = gcdCondit
  auto gcd2 = gcdCommon(100, 10LL);

  std::cout << std::endl;

  std::cout << "typeid(gcd1).name() = " << typeid(gcd1).name() << std::endl;
  std::cout << "typeid(gcd2).name() = " << typeid(gcd2).name() << std::endl;

  std::cout << std::endl;

}
```



We'll look at the solution review of this exercise in the next lesson.