- Solution

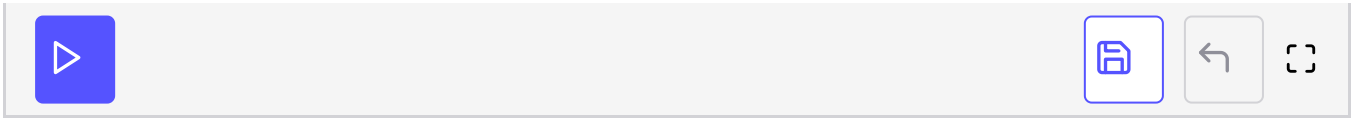
In this lesson, we'll look at the solution review of the previous challenge.

WE'LL COVER THE FOLLOWING

Solution Review

```
#include <cstddef>
#include <iostream>
#include <typeinfo>

template <char c>
class AcceptChar{
public:
  AcceptChar(){
    std::cout << "AcceptChar: "  << typeid(c).name() << std::endl;
  }
};

template < int(*func)(int) >
class AcceptFunction{
public:
  AcceptFunction(){
    std::cout << "AcceptFunction: "  << typeid(func).name() << std::endl;
  }
};

template < int(&arr)[5] >
class AcceptReference{
public:
  AcceptReference(){
    std::cout << "AcceptReference: " << typeid(arr).name() << std::endl;
  }
};

template < std::nullptr_t N >
class AcceptNullptr{
public:
  AcceptNullptr(){
    std::cout << "AcceptNullpt: " << typeid(N).name() << std::endl;
  }
};
```

```
int myFunc(int){ return 2011; };
int arr[5];


int main(){

  std::cout << std::endl;

  AcceptChar<'c'> acceptChar;
  AcceptFunction< myFunc> acceptFunction;
  AcceptReference< arr > acceptReference;
  AcceptNullptr< nullptr > acceptNullptr;

  std::cout << std::endl;

}
```



Explanation

We have created four different class templates which include `AcceptChar`, `AcceptFunction`, `AcceptReference`, and `AcceptNull` in lines 6, 14, 22, and 30. Each class template accepts a different non-type. To verify all, we have declared a *character* variable, a *reference* to an array, a function, and `nullptr` in lines 44 – 47. This matches the order of declaration in the code. We identify their type using the operator `typeid` in lines 9, 17, 25, and 33.

Let's move on to template arguments in the next lesson.