

- Solution

Let's look at the solution review for the exercise we presented in the last lesson.

WE'LL COVER THE FOLLOWING ^

- Solution Review
- Explanation

Solution Review

```
// Template Types

#include <iostream>
#include <string>

class Account{};

template<typename T>
struct Type{
    std::string getName() const {
        return "unknown";
    }
};

template<typename T>
struct Type<T*>{
    std::string getName() const {
        return "pointer";
    }
};

template<typename T>
struct Type<const T>{
    std::string getName() const {
        return "const";
    }
};

template<>
struct Type<int>{
    std::string getName() const {
        return "int";
    }
};
```



```

template<>
struct Type<double>{

    std::string getName() const {
        return "double";
    }
};

template<>
struct Type<std::string>{
    std::string getName() const {
        return "std::string";
    }
};

template<>
struct Type<Account>{
    std::string getName() const {
        return "Account";
    }
};

int main(){

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

    Type<float> tFloat;
    std::cout << "tFloat.getName(): " << tFloat.getName() << std::endl;

    Type<const float> tConstFloat;
    std::cout << "tConstFloat.getName(): " << tConstFloat.getName() << std::endl;

    Type<float*> tFloatPointer;
    std::cout << "tFloatPointer.getName(): " << tFloatPointer.getName() << std::endl;

    Type<double> tDouble;
    std::cout << "tDouble.getName(): " << tDouble.getName() << std::endl;

    Type<std::string> tString;
    std::cout << "tString.getName(): " << tString.getName() << std::endl;

    Type<int> tInt;
    std::cout << "tInt.getName(): " << tInt.getName() << std::endl;

    Type<Account> tAccount;
    std::cout << "tAccount.getName(): " << tAccount.getName() << std::endl;

    std::cout << std::endl;

}

```



Explanation

In the above code, we have separately defined the partial and full specialization of the class template **Type**. The partial and full specializations

accept an `int`, `double`, `Account`, `string`, `const`, and `Pointer`. On calling each type, the relative type is returned. We have not defined the full specialization for `float`, so when instantiating the class template for `float`, it gives an `unknown` in response.

We have covered the basics of templates in this chapter. In the next chapter, we'll learn about the details of templates. Let's start with template instantiation in the next lesson.