

- Solution

Let's have a look at the solution of the last exercise.

WE'LL COVER THE FOLLOWING ^

- Solution Review
- Explanation

Solution Review

```
// TypeErasureTypeInfo.cpp

#include <iostream>
#include <memory>
#include <string>
#include <typeinfo>
#include <vector>

struct Object{
    std::string getTypeName() const { return _inner->getTypeName(); }
    struct Concept{
        using ptr = std::unique_ptr<Concept>;
        virtual std::string getTypeName() const = 0;
    };
    template <typename T>
    struct Model : Concept{
        std::string getTypeName() const override { return typeid(T).name(); }
    };
private:
    typename Concept::ptr _inner;
};

template <>
struct Object::Model<long long> : Object::Concept{
    std::string getTypeName() const override { return "long long"; }
};

template <>
struct Object::Model<std::string> : Object::Concept{
    std::string getTypeName() const override { return "std::string<char>"; }
};

template <>
struct Object::Model<std::vector<int>> : Object::Concept{
    std::string getTypeName() const override { return "std::vector<int>"; }
```

```

};

template <typename T>
void printType(T&& t){
    std::cout << t.getTypeName() << std::endl;
}

class Test{};

int main(){

    printType(Object::Model<int>{});
    printType(Object::Model<double>{});
    printType(Object::Model<void>{});
    printType(Object::Model<Test>{});
    printType(Object::Model<Object>{});
    printType(Object::Model<long long>{});
    printType(Object::Model<std::string>{});
    printType(Object::Model<std::vector<int>>{});

}

```



Explanation

The function template `printType` returns a string representation for each type. In the general case, the primary template `Model` (lines 15 – 18) is used. The implementation uses the `typeid` operator to get the name. Full specializations are also available for the types `long long` (lines 23 – 26), `std::string` (lines 28 – 31), and `int` (lines 33 – 36). The full specializations provide personalized string representations.

In this chapter, we have learned about design techniques of C++ templates. In the next chapter, we'll look at the future concepts in C++20.