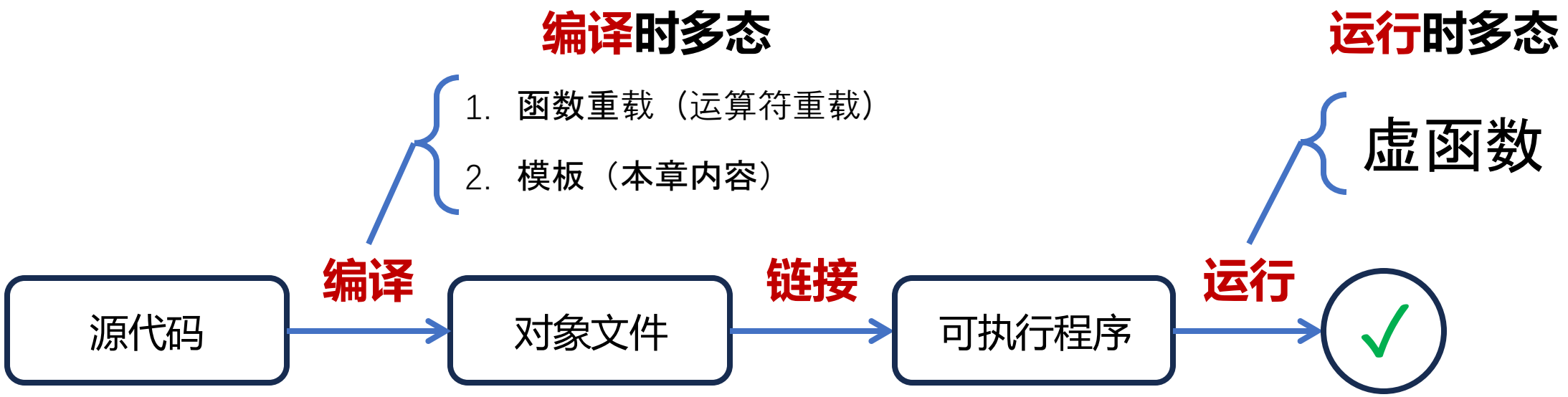
模板

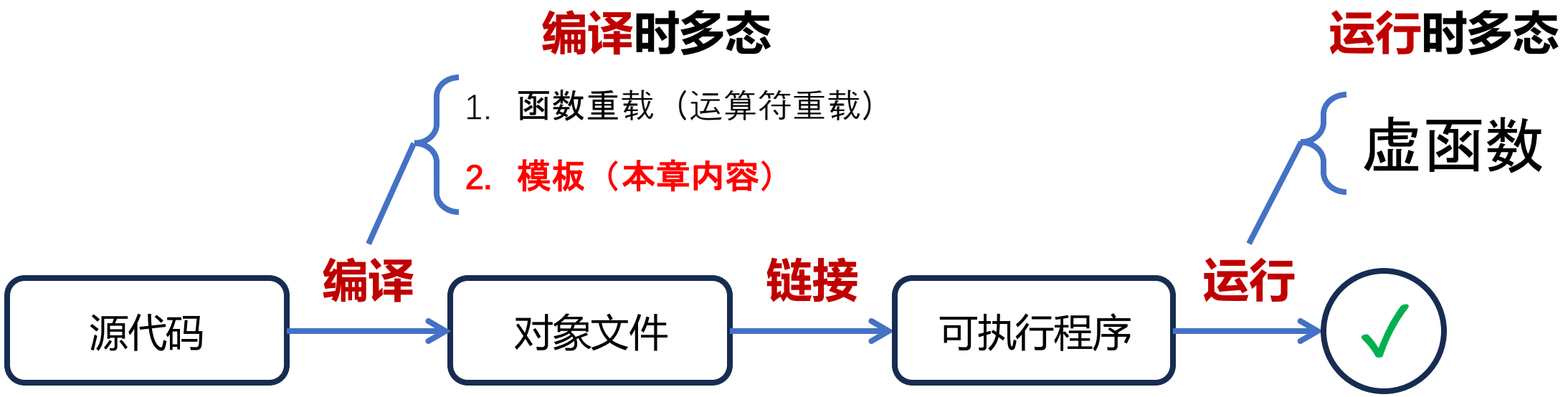
胡船长

初航我带你，远航靠自己

基础概念：多态的分类



基础概念：多态的分类



编程范式的对比

	C 语言	C++ 语言
面向过程编程	✅	✅
面向对象编程	❌	✅
泛型编程	❌	✅
函数式编程	❌	✅

编程范式的对比

	C 语言	C++ 语言
面向过程编程	✅	✅
面向对象编程	❌	✅
泛型编程	❌	✅
函数式编程	❌	✅

程序 = 算法 + 数据结构

数据结构： 能够存储**任意类型**

算　　法： 能够操作存储**任意类型**数据的数据结构

泛型编程

将【**任意类型**】从**程序设计**中抽象出来

泛型编程

	泛型编程
面向过程编程	用 **模板** 实现函数过程
面向对象编程	用 **模板** 实现类

模板基础

1. 语法基础：模板函数
2. 趁热打铁：实现一个完美的 add 模板
3. 深入探索：模板的类型推导机制
4. 语法基础：模板类
5. 全特化与偏特化

变参模板

1. 语法基础：变参模板
2. 编码技巧：解析变参列表
3. 学以致用：实现 function 模板类

模板元编程：初体验

1. 功能1：判断偶数
2. 功能2：累加求和
3. 功能3：判断素数
4. 功能4：素数累加求和

项目实战

1. 复现 STL：类型转换模板
2. 复现 STL：vector 模板类
3. 复现 STL：map 模板类
4. 特化实战：bool 类型的 vector
5. 灵机一动：统计函数执行次数
6. 综合实战：实现 thread_pool 线程池

模板基础

1. **语法基础：模板函数**
2. 趁热打铁：实现一个完美的 add 模板
3. 深入探索：模板的类型推导机制
4. 语法基础：模板类
5. 全特化与偏特化

语法基础：模板

模板函数

```
template<typename T>
T add(T a, T b) {
    return a + b;
}
```

模板类

```
template<typename T>
struct PrintAny {
    PrintAny(std::ostream &out) : out(out) {}
    void operator()(const T &a) {
        out << a;
    }
    std::ostream &out;
};
```

模板基础

1. 语法基础：模板函数
2. **趁热打铁：实现一个完美的 add 模板**
3. 深入探索：模板的类型推导机制
4. 语法基础：模板类
5. 全特化与偏特化

模板基础

1. 语法基础：模板函数
2. 趁热打铁：实现一个完美的 add 模板
3. **深入探索：模板的类型推导机制**
4. 语法基础：模板类
5. 全特化与偏特化

模板的类型推导

- 隐式推导
- 显示推导
- 间接推导
- 引用类型的推导

模板的类型推导

- ☑ 隐式推导
- ☐ 显示推导
- ☐ 间接推导
- ☐ 引用类型的推导

模板的类型推导

- ☑ 隐式推导
- ☑ 显示推导
- ☐ 间接推导
- ☐ 引用类型的推导

模板的类型推导

- [x] 隐式推导
- [x] 显示推导
- [x] 间接推导
- [ ] 引用类型的推导

模板的类型推导

- ☑ 隐式推导
- ☑ 显示推导
- ☑ 间接推导
- ☑ 引用类型的推导

模板基础

1. 语法基础：模板函数
2. 趁热打铁：实现一个完美的 add 模板
3. 深入探索：模板的类型推导机制
4. **语法基础：模板类**
5. 全特化与偏特化

语法基础：模板

模板函数

```
template<typename T>
T add(T a, T b) {
    return a + b;
}
```

模板类

```
template<typename T>
struct PrintAny {
    PrintAny(std::ostream &out) : out(out) {}
    void operator()(const T &a) {
        out << a;
    }
    std::ostream &out;
};
```

语法基础：模板

模板类 + 模板函数

```
template<typename T>
struct Print{
    template<typename U>
    void operator()(const U &a) {
        cout << a << endl;
        cout << this->__temp << endl;
    }
    void set(const T &temp) {this->__temp = temp;}
    T __temp;
};
```

模板基础

1. 语法基础：模板函数
2. 趁热打铁：实现一个完美的 add 模板
3. 深入探索：模板的类型推导机制
4. 语法基础：模板类
5. **全特化与偏特化**

模板的全特化

模板函数

```
template<typename T>
T add(T a, T b) {
    return a + b;
}
```

特化

模板函数特化

```
template<>
int add(int a, int b) {
    return a + b + 2;
}
```

模板的全特化

模板类特化

```
template<typename T>
struct PrintAny {
    PrintAny(std::ostream &out) : out(out) {}
    void operator()(const T &a) {
        out << a;
    }
    std::ostream &out;
};
```

模板的全特化

模板类特化

```
template<>
struct PrintAny<int> {
    PrintAny(std::ostream &out) : out(out) {}
    template<typename U>
    void operator[](const U &a) {
        out << a;
    }
    std::ostream &out;
};
```

模板的偏特化

模板函数

```
template<typename T>
void P(T a) {
    cout << a << endl;
}
```

偏特化

模板函数偏特化

```
template<typename T>
void P(T *a) {
    cout << *a << endl;
}
```

变参模板

1. **语法基础：变参模板**
2. 编码技巧：解析变参列表
3. 学以致用：实现 function 模板类

可变参数模板

```
template<typename T, typename ...ARGS>
void Print(const T &a, ARGS... args) {
    cout << a << endl;
    Print(args...);
}
```

代码讲解：

ARGS 代表模板中剩余部分的类型数量是可变的，但最少为 1 个。

此代码会递归展开模板函数 Print

可变参数模板

递归展开模板函数

```
template<typename T, typename ...ARGS>
void Print(const T &a, ARGS... args) {
    cout << a << endl;
    Print(args...);
}
```

终止递归展开

```
template<typename T>
void Print(const T &a) {
    cout << a << endl;
}
```

变参模板

1. 语法基础：变参模板
2. **编码技巧：解析变参列表**
3. 学以致用：实现 function 模板类

可变参数模板

```
template<typename T, typename ...REST>
struct ARG {
    typedef T __type;
    typedef ARG<REST...> __rest;
};
```

```
template<typename T>
struct ARG<T> {
    typedef T __type;
};
```

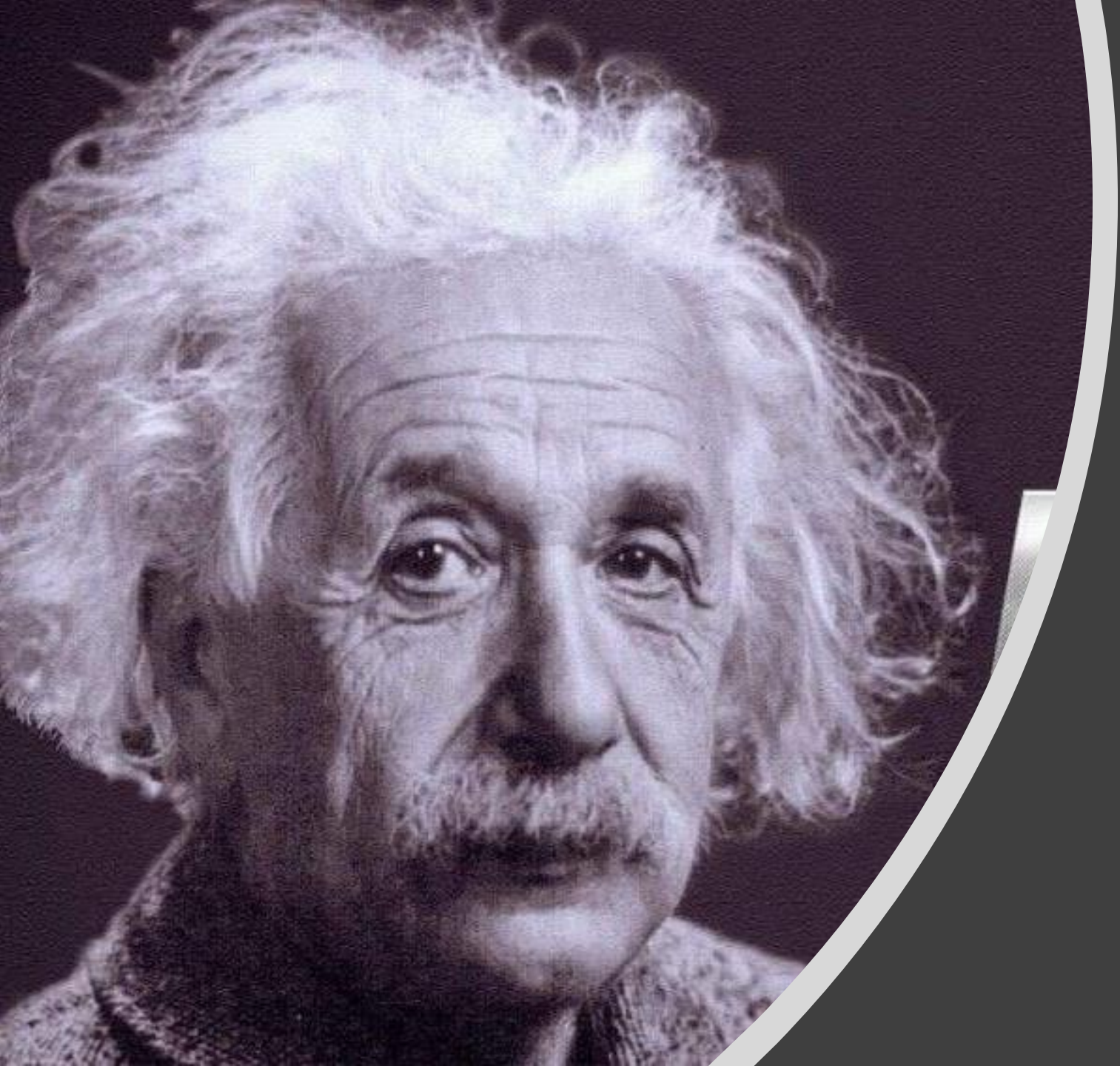
```
template<typename T, typename ...ARGS> struct Test;
template<typename T, typename ...ARGS>
struct Test<T(ARGS...)> {
    T operator()(typename ARG<ARGS...>::__type a, typename ARG<ARGS...>::__rest::__type b) {
        return a + b;
    }
};
```

课后思考题

如何使得上面的Test 模板类，REST...参数部分有且只有**两个**参数

变参模板

1. 语法基础：变参模板
2. 编码技巧：解析变参列表
3. **学以致用：实现 function 模板类**



为什么

会出一样的题目？