电子科技大学

计算机专业类课程

实验报告

课程名称： C++语言程序设计

学院专业： 计算机科学与工程学院

学生姓名： 何阳

学　　号： 2024080903004

指导教师： 郭磊

日　　期： 2025 年 11 月 21 日

电 子 科 技 大 学

实 验 报 告

实验三

一、实验室名称：

电子科技大学清水河校区主楼 A2-413-1

二、实验项目名称：

C++智能车项目组件

三、实验目的：

1. 掌握表驱动设计模式，优化条件判断逻辑，提升代码扩展性。
2. 学会通过数据抽离和解耦循环依赖，改善代码结构。
3. 理解函数式编程思想，掌握 Lambda 表达式和操作符重载的使用。
4. 提升代码的模块化和安全性，确保对象生命周期管理的正确性。

四、实验器材（设备、元器件）：

操作系统：Windows11
编译程序：Visual Studio Code
编译器：MinGW64
操作系统：Windows 10
开发工具：Visual Studio Code、Git
测试框架：Google Test（gtest）

五、实验主要内容：

（一）解耦循环依赖

1. 分析 `ExecutorImpl` 与 `ICommand` 的循环依赖问题，抽离状态数据到 `PoseHandler` 类。
2. `PoseHandler` 封装位置、朝向、状态标记及相关操作（`Move`、`TurnLeft`、`Fast`），`Command` 子类依赖 `PoseHandler` 而非 `ExecutorImpl`，消除循环依赖。

（二）表驱动优化

1. 使用 `std::unordered_map` 创建指令-命令对象映射表，key 为指令字符（M/L/R/F），value 为对应 `Command` 对象。
2. 重构 `ExecutorImpl::Execute` 方法，通过查表替代条件判断，简化指令分发逻辑。

（三）函数式编程改造

1. 删除 `ICommand` 抽象类，通过 `std::function` 封装指令行为，结合 Lambda 表达式简化代码。
2. 为 `Command` 类重载 `operator()`，将函数行为封装为对象，确保生命周期安全。

（四）倒车指令（B）扩展

1. 基于表驱动和函数式编程思想，新增 `ReverseCommand`，实现倒车状态切换。
2. 扩展 `PoseHandler` 支持倒车状态下的指令行为（如 M 指令后退，L/R 指令转向反转）。

六、实验结果

1. 成功解耦循环依赖，代码结构更清晰，模块职责单一。
2. 表驱动优化后，新增指令无需修改 `Execute` 核心逻辑，扩展性显著提升。
3. 函数式编程改造使代码更简洁，Lambda 表达式减少了冗余代码。
4. 倒车指令（B）功能正常实现，支持与 F 指令叠加，所有测试用例通过。
5. Gtest 截图如下

```
PS C:\Users\Suuny\Desktop\repository\C++作业\build\debug> .\test_executor.exe
[==========] Running 20 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 20 tests from ExecutorTest
[ RUN      ] ExecutorTest.DefaultInitialization
[       OK ] ExecutorTest.DefaultInitialization (0 ms)
[ RUN      ] ExecutorTest.CustomInitialization
[       OK ] ExecutorTest.CustomInitialization (0 ms)
[ RUN      ] ExecutorTest.MoveForwardCommand
[       OK ] ExecutorTest.MoveForwardCommand (0 ms)
[ RUN      ] ExecutorTest.TurnLeftCommand
[       OK ] ExecutorTest.TurnLeftCommand (0 ms)
[ RUN      ] ExecutorTest.TurnRightCommand
[       OK ] ExecutorTest.TurnRightCommand (0 ms)
[ RUN      ] ExecutorTest.BatchCommandsExecution
[       OK ] ExecutorTest.BatchCommandsExecution (0 ms)
[ RUN      ] ExecutorTest.GetStatusMethod
[       OK ] ExecutorTest.GetStatusMethod (0 ms)
[ RUN      ] ExecutorTest.AccelerationToggle
[       OK ] ExecutorTest.AccelerationToggle (0 ms)
[ RUN      ] ExecutorTest.AccelerationMoveForward
[       OK ] ExecutorTest.AccelerationMoveForward (0 ms)
[ RUN      ] ExecutorTest.AccelerationTurnLeft
[       OK ] ExecutorTest.AccelerationTurnLeft (0 ms)
[ RUN      ] ExecutorTest.AccelerationTurnRight
[       OK ] ExecutorTest.AccelerationTurnRight (0 ms)
[ RUN      ] ExecutorTest.AccelerationComplex
[       OK ] ExecutorTest.AccelerationComplex (0 ms)
[ RUN      ] ExecutorTest.ReverseToggle
[       OK ] ExecutorTest.ReverseToggle (0 ms)
[ RUN      ] ExecutorTest.ReverseModeCommands
[       OK ] ExecutorTest.ReverseModeCommands (0 ms)
[ RUN      ] ExecutorTest.Combined_F_B_State_Commands
[       OK ] ExecutorTest.Combined_F_B_State_Commands (0 ms)
[ RUN      ] ExecutorTest.CombinedStateComplex
[       OK ] ExecutorTest.CombinedStateComplex (0 ms)
[ RUN      ] ExecutorTest.TurnRoundNormal
[       OK ] ExecutorTest.TurnRoundNormal (0 ms)
[ RUN      ] ExecutorTest.TurnRoundAccelerated
[       OK ] ExecutorTest.TurnRoundAccelerated (0 ms)
[ RUN      ] ExecutorTest.TurnRoundUnaffectedByReverse
[       OK ] ExecutorTest.TurnRoundUnaffectedByReverse (0 ms)
[ RUN      ] ExecutorTest.TurnRoundComplex
[       OK ] ExecutorTest.TurnRoundComplex (0 ms)
[----------] 20 tests from ExecutorTest (4 ms total)

[----------] Global test environment tear-down
[==========] 20 tests from 1 test suite ran. (5 ms total)
[  PASSED  ] 20 tests.
```