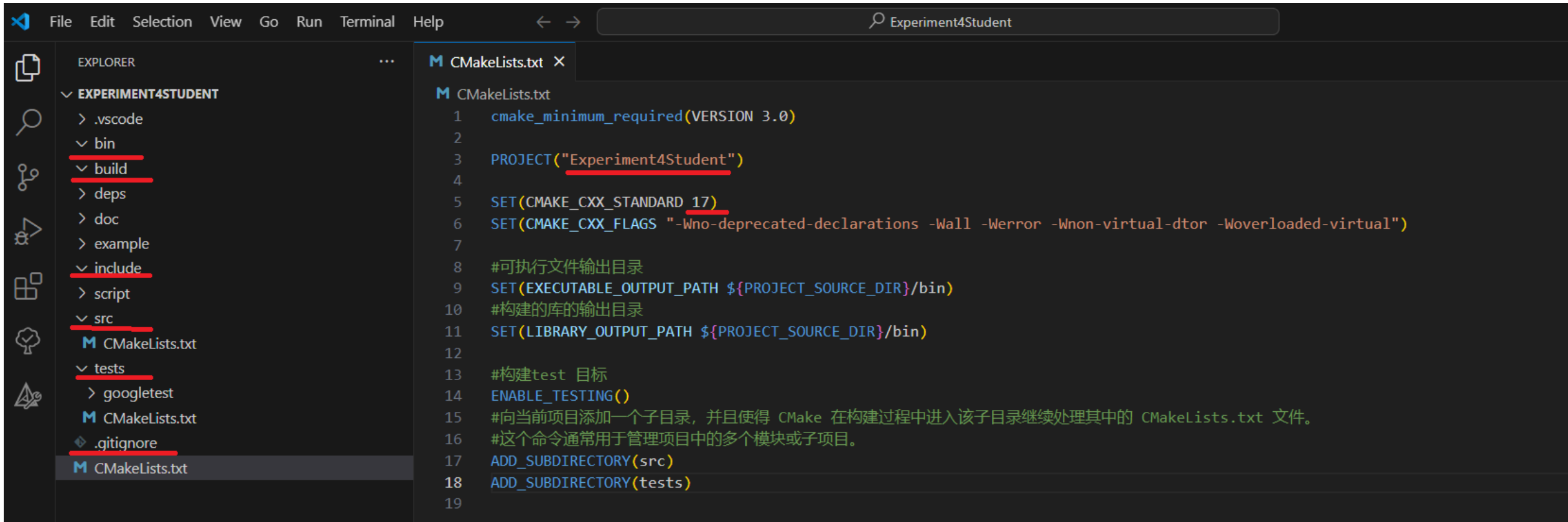


实验—StepbyStep

从工程模版创建一个空的工程， 比如叫Experiment4Student



首先可以看到工程下的bin、build、include目录都是空的，src和tests目录下没有任何C++源码文件

请修改根目录下的CMakeLists.txt(红色下划线所示，C++标准改成17，工程名称改成项目根目录名)

同时要看到工程根目录下有一个.gitignore文件，这个是配置当提交代码到git时，哪些文件/目录是不需要提交的。在更新的工程模板里已经添加了这个文件

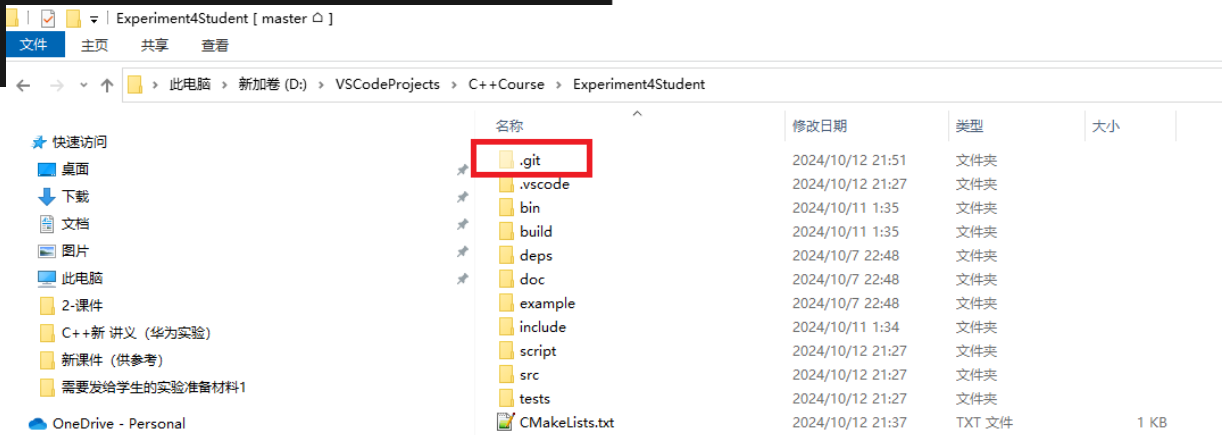
将这个工程提交给本地git仓库（因为暂时没有远程服务器搭建远程仓库）

打开Terminal，输入下面命令创建本地仓库

git init #初始化仓库可以发现当前目录下多了一个.git的目录，这个就是本地仓库

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001
PS D:\VSCodeProjects\C++Course\Experiment4Student> git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint: git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint: git branch -m <name>
Initialized empty Git repository in D:/VSCodeProjects/C++Course/Experiment4Student/.git/
PS D:\VSCodeProjects\C++Course\Experiment4Student>
```



将这个工程提交给本地git仓库（因为暂时没有远程服务器搭建远程仓库）

打开Terminal，输入下面命令创建本地仓库

git init #初始化仓库可以发现当前目录下多了一个.git的目录，这个就是本地仓库

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001
PS D:\VSCodeProjects\C++Course\Experiment4Student> git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint: git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint: git branch -m <name>
Initialized empty Git repository in D:/VSCodeProjects/C++Course/Experiment4Student/.git/
PS D:\VSCodeProjects\C++Course\Experiment4Student>

Experiment4Student [master]

文件 主页 共享 查看

此电脑 > 新加坡 (D:) > VSCodeProjects > C++Course > Experiment4Student

| 名称 | 修改日期 | 类型 | 大小 |
|----------------|------------------|--------|------|
| .git | 2024/10/12 21:51 | 文件夹 | |
| .vscode | 2024/10/12 21:27 | 文件夹 | |
| bin | 2024/10/11 1:35 | 文件夹 | |
| build | 2024/10/11 1:35 | 文件夹 | |
| deps | 2024/10/7 22:48 | 文件夹 | |
| doc | 2024/10/7 22:48 | 文件夹 | |
| example | 2024/10/7 22:48 | 文件夹 | |
| include | 2024/10/11 1:34 | 文件夹 | |
| script | 2024/10/12 21:27 | 文件夹 | |
| src | 2024/10/12 21:27 | 文件夹 | |
| tests | 2024/10/12 21:27 | 文件夹 | |
| CMakeLists.txt | 2024/10/12 21:37 | TXT 文件 | 1 KB |

将这个工程提交给本地git仓库（因为暂时没有远程服务器搭建远程仓库）

在Terminal里接着输入下面二行命令

```
git add .
```

#提交修改过或新增的文件到暂存区

```
git commit -m "init project"
```

#将暂存区的文件提交到本地仓库中

注意：每次提交代码到本地仓库时，一定要规范-m 后面的comment信息

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> git add .
```

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> git commit -m "init project"
```

```
[master (root-commit) 3c692e2] init project
```

```
7 files changed, 161 insertions(+)
```

```
create mode 100644 .gitignore
```

```
create mode 100644 CMakeLists.txt
```

```
create mode 100644 script/build.bat
```

```
create mode 100644 script/clean.bat
```

```
create mode 100644 script/run.bat
```

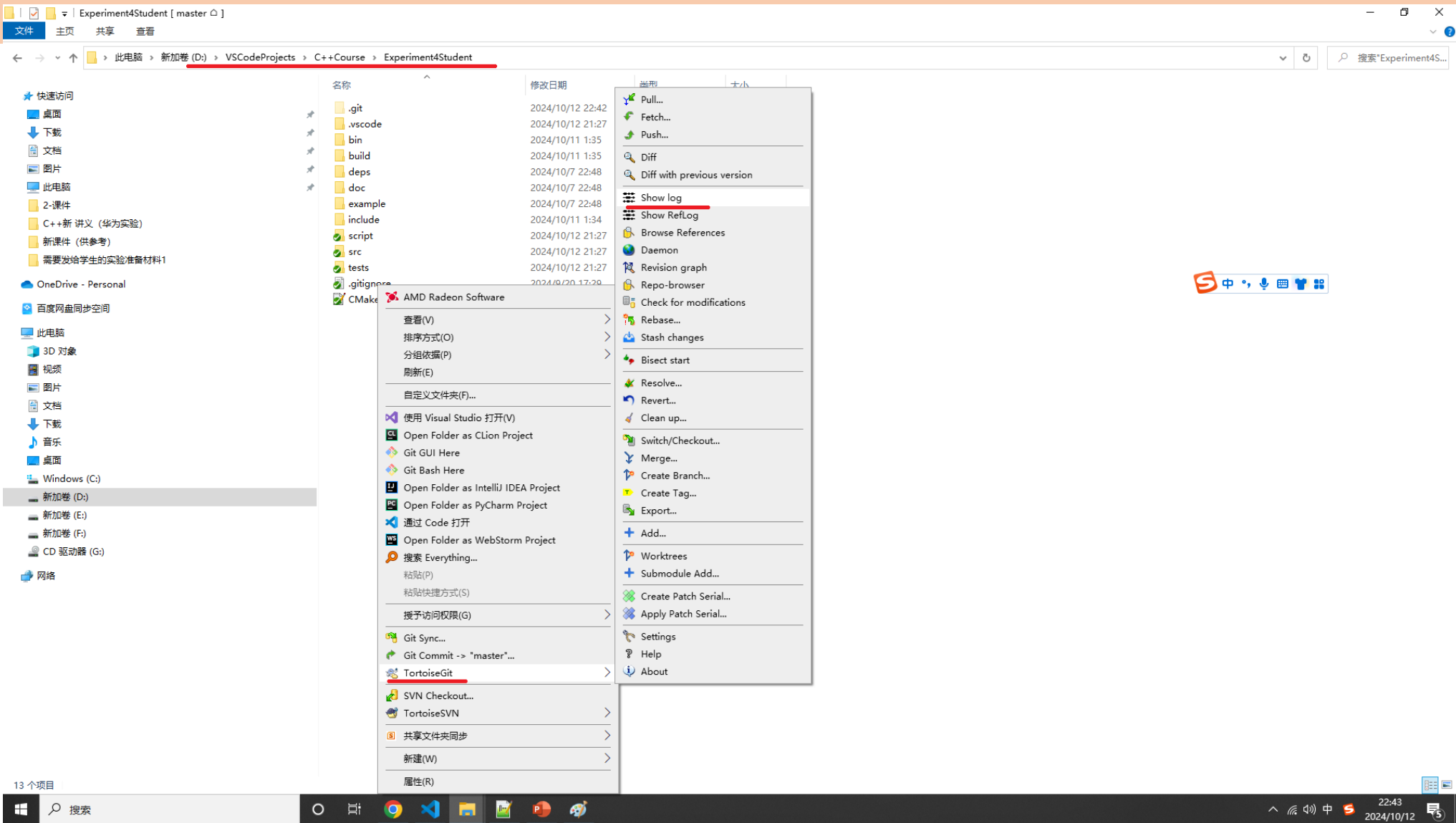
```
create mode 100644 src/CMakeLists.txt
```

```
create mode 100644 tests/CMakeLists.txt
```

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> █
```

建议安装TortoiseGit (<https://tortoisegit.org/download/>)

安装过程一路默认就行。安装好，打开工程目录，里面应该可以看到.git子目录。在空白处点击鼠标右键



通过TortoiseGit查看提交的log日志

D:\VSCodeProjects\C++Course\Experiment4Student - Log Messages - TortoiseGit

master

From: 2024/10/12 To: 2024/10/12

Filter by Subject, Messages, Paths, Authors, Emails, SHA-1, Refname, Notes

Author Email

| Graph | Actions | Message | Author | Date |
|-------|---------|--|--------|---------------------|
| | | Working tree changes master init project | guxiwu | 2024/10/12 22:18:05 |

SHA-1: 3c692e290439961b98c180c559ba89ec33d5adc8

* init project

| Path | Extension | Status | Lines added | Lines removed |
|----------------------|------------|--------|-------------|---------------|
| .gitignore | .gitignore | Added | 25 | 0 |
| CMakeLists.txt | .txt | Added | 18 | 0 |
| script/build.bat | .bat | Added | 13 | 0 |
| script/clean.bat | .bat | Added | 24 | 0 |
| script/run.bat | .bat | Added | 25 | 0 |
| src/CMakeLists.txt | .txt | Added | 32 | 0 |
| tests/CMakeLists.txt | .txt | Added | 24 | 0 |

Showing 1 revision(s), from revision 3c692e29 to revision 3c692e29 - 1 revision(s) selected, 0 file(s) selected; line: 161(+)-0(-) files: modified = 0 added = 7 deleted = 0 replaced = 0

☐ Show Whole Project

☐ All Branches

Filter paths

Help

Refresh

Statistics

Walk Behavior

View

OK

Windows taskbar with search bar and system tray showing time 22:50 and date 2024/10/12.

在include下添加Executor.hpp

VScode里新添加的文件绿色显示，表示还没提交到本地仓库

实验所有代码都位于名字空间adas

Pose描述汽车姿态

动作执行接口Executor
由于Query是纯虚函数，所以Executor是抽象类，无法实例化。
另外有一个静态方法NewExecutor，会返回Executor接口类型指针Executor*，这是一个工厂方法。
另外还要注意Executor没有定义任何数据成员。

纯虚函数Query、静态方法NewExecutor、数据成员全部放到Executor的子类ExecutorImpl去实现。

```
include > Executor.hpp > {} adas > Executor > Query(void) const
1  #pragma once
2  #include <string>
3  namespace adas
4  {
5      /* 汽车姿态 */
6      struct Pose
7      {
8          int x;          //x坐标
9          int y;          //y坐标
10         char heading;    //'N','S','E','W'代表四个方向
11     };
12
13     /*
14     驾驶动作执行器接口
15     */
16     class Executor
17     {
18     public:
19         // Caller should delete *executor when it is no longer needed.
20         static Executor *NewExecutor(const Pose &pose = {0, 0, 'N'}) noexcept;
21
22     public:
23         //默认构造函数
24         Executor(void) = default;
25         //默认虚析构函数
26         virtual ~Executor(void) = default;
27
28         //不允许拷贝
29         Executor(const Executor &) = delete;
30         //不允许赋值
31         Executor &operator=(const Executor &) = delete;
32
33     public:
34         //查询当前汽车姿态，纯虚函数，留给子类具体实现
35         virtual Pose Query(void) const noexcept = 0;
36     };
37 } // namespace adas
```


在tests下添加ExecutorTest.cpp

TEST是一个宏，包括二个参数：

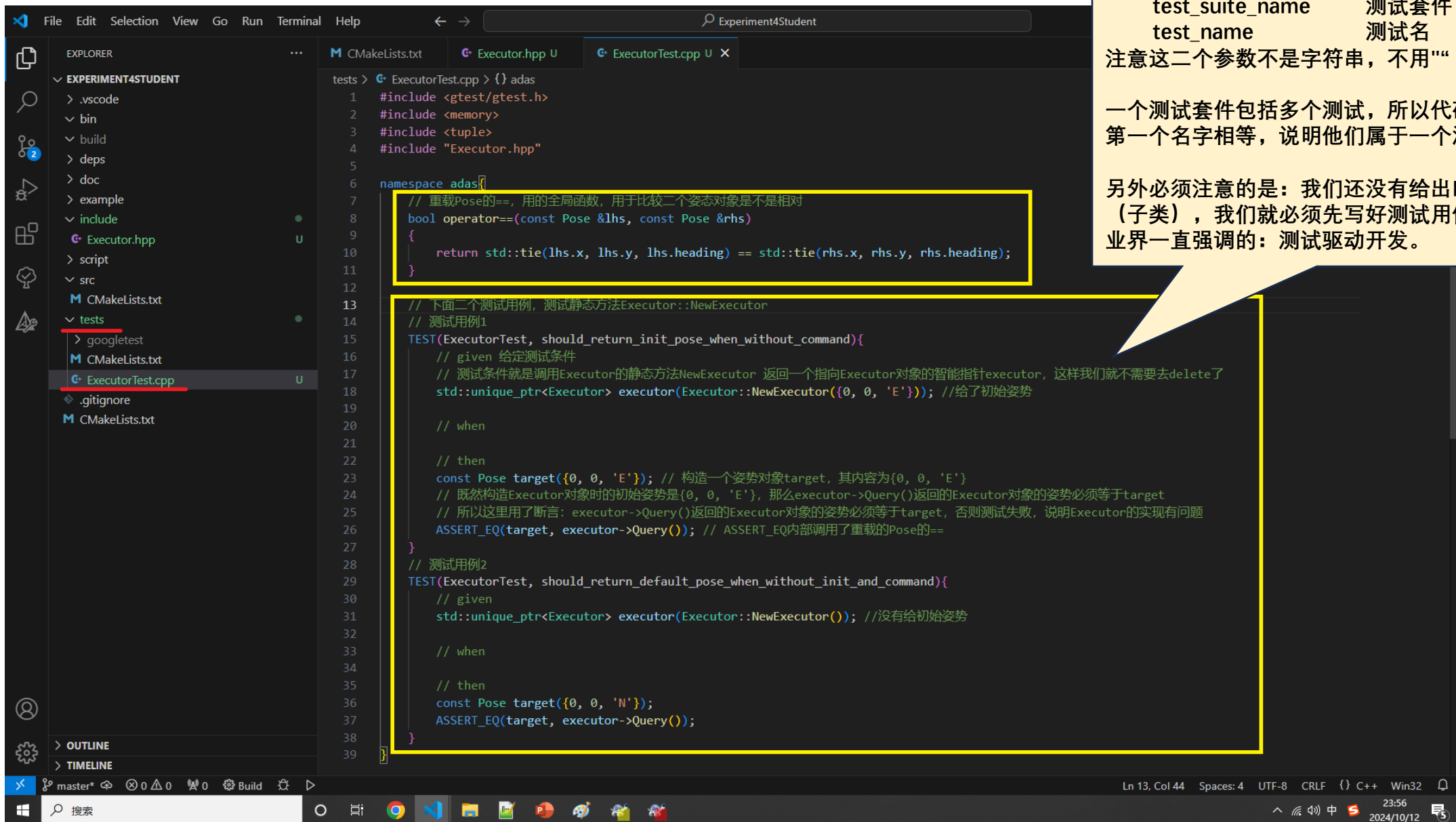
test_suite_name 测试套件

test_name 测试名

注意这二个参数不是字符串，不用""

一个测试套件包括多个测试，所以代码里二个测试的第一个名字相等，说明他们属于一个测试套件

另外必须注意的是：我们还没有给出Executor的实现（子类），我们就必须先写好测试用例代码。这就是业界一直强调的：测试驱动开发。



```
1 #include <gtest/gtest.h>
2 #include <memory>
3 #include <tuple>
4 #include "Executor.hpp"
5
6 namespace adas{
7     // 重载Pose的==, 用的全局函数, 用于比较二个姿态对象是不是相对
8     bool operator==(const Pose &lhs, const Pose &rhs)
9     {
10         return std::tie(lhs.x, lhs.y, lhs.heading) == std::tie(rhs.x, rhs.y, rhs.heading);
11     }
12
13     // 下面二个测试用例, 测试静态方法Executor::NewExecutor
14     // 测试用例1
15     TEST(ExecutorTest, should_return_init_pose_when_without_command){
16         // given 给定测试条件
17         // 测试条件就是调用Executor的静态方法NewExecutor 返回一个指向Executor对象的智能指针executor, 这样我们就不需要去delete了
18         std::unique_ptr<Executor> executor(Executor::NewExecutor({0, 0, 'E'})); //给了初始姿势
19
20         // when
21
22         // then
23         const Pose target({0, 0, 'E'}); // 构造一个姿态对象target, 其内容为{0, 0, 'E'}
24         // 既然构造Executor对象时的初始姿势是{0, 0, 'E'}, 那么executor->Query()返回的Executor对象的姿势必须等于target
25         // 所以这里用了断言: executor->Query()返回的Executor对象的姿势必须等于target, 否则测试失败, 说明Executor的实现有问题
26         ASSERT_EQ(target, executor->Query()); // ASSERT_EQ内部调用了重载的Pose的==
27     }
28     // 测试用例2
29     TEST(ExecutorTest, should_return_default_pose_when_without_init_and_command){
30         // given
31         std::unique_ptr<Executor> executor(Executor::NewExecutor()); //没有给初始姿势
32
33         // when
34
35         // then
36         const Pose target({0, 0, 'N'});
37         ASSERT_EQ(target, executor->Query());
38     }
39 }
```

在src下添加ExecutorImpl.hpp

The screenshot shows the Visual Studio Code editor with the file `ExecutorImpl.hpp` open in the `src` directory. The code defines a class `ExecutorImpl` that inherits from `Executor`. Annotations explain key parts of the implementation:

- ExecutorImpl从Executor公有派生**: Points to the `class ExecutorImpl: public Executor` line, indicating public inheritance.
- 要实现父类的纯虚函数Query, 所以后面加了override**: Points to the `Pose Query(void) const noexcept override;` line, explaining the use of `override` for a pure virtual function.
- 在ExecutorImpl里面就定义私有的数据成员pose, 记录执行器当前的姿势**: Points to the `private: Pose pose;` line, explaining the private data member used to store the current pose.

```
1  #pragma once
2
3  #include "Executor.hpp"
4  #include <string>
5
6  namespace adas{
7      /*
8       * Executor的具体实现
9       */
10     class ExecutorImpl: public Executor{
11     public:
12         //构造函数
13         explicit ExecutorImpl(const Pose& pose) noexcept;
14         //默认析构函数
15         ~ExecutorImpl() noexcept = default;
16
17         //不能拷贝
18         ExecutorImpl(const ExecutorImpl &) = delete;
19         //不能赋值
20         ExecutorImpl &operator=(const ExecutorImpl &) = delete;
21
22     public:
23         // 查询当前汽车姿态, 是父类抽象方法Query的具体实现
24         Pose Query(void) const noexcept override;
25
26     private:
27         //私有数据成员, 汽车当前姿态
28         Pose pose;
29     };
30 }
```

在src下添加ExecutorImpl.cpp

The screenshot shows the Visual Studio Code editor with the file Explorer on the left displaying the project structure. The Explorer shows a folder named 'EXPERIMENT4STUDENT' containing subfolders like '.vscode', 'bin', 'build', 'deps', 'doc', 'example', 'include', 'script', and 'src'. The 'src' folder is expanded, showing files like 'CMakeLists.txt', 'ExecutorImpl.cpp' (highlighted), 'ExecutorImpl.hpp', 'tests', 'googletest', 'CMakeLists.txt', 'ExecutorTest.cpp', '.gitignore', and 'CMakeLists.txt'.

The main editor window displays the code for 'ExecutorImpl.cpp'. The code is as follows:

```
src > ExecutorImpl.cpp > {} adas > NewExecutor(const Pose &)  
1 #include "ExecutorImpl.hpp"  
2  
3 #include <new>  
4  
5 namespace adas{  
6     ExecutorImpl::ExecutorImpl(const Pose &pose) noexcept {}  
7  
8     Pose ExecutorImpl::Query(void) const noexcept{  
9         return pose;  
10    }  
11  
12    /*  
13     std::nothrow 是 C++ 标准库中的一个常量，用于指示在分配内存时不抛出任何异常。  
14     它是 std::nothrow_t 类型的实例，通常用在 new 运算符和 std::nothrow 命名空间中，  
15     以请求内存分配器在分配失败时返回一个空指针，而不是抛出 std::bad_alloc 异常。  
16    */  
17    Executor *Executor::NewExecutor(const Pose &pose) noexcept{  
18        return new(std::nothrow) ExecutorImpl(pose); //只在C++17下有效  
19    }  
20  
21  
22 }
```

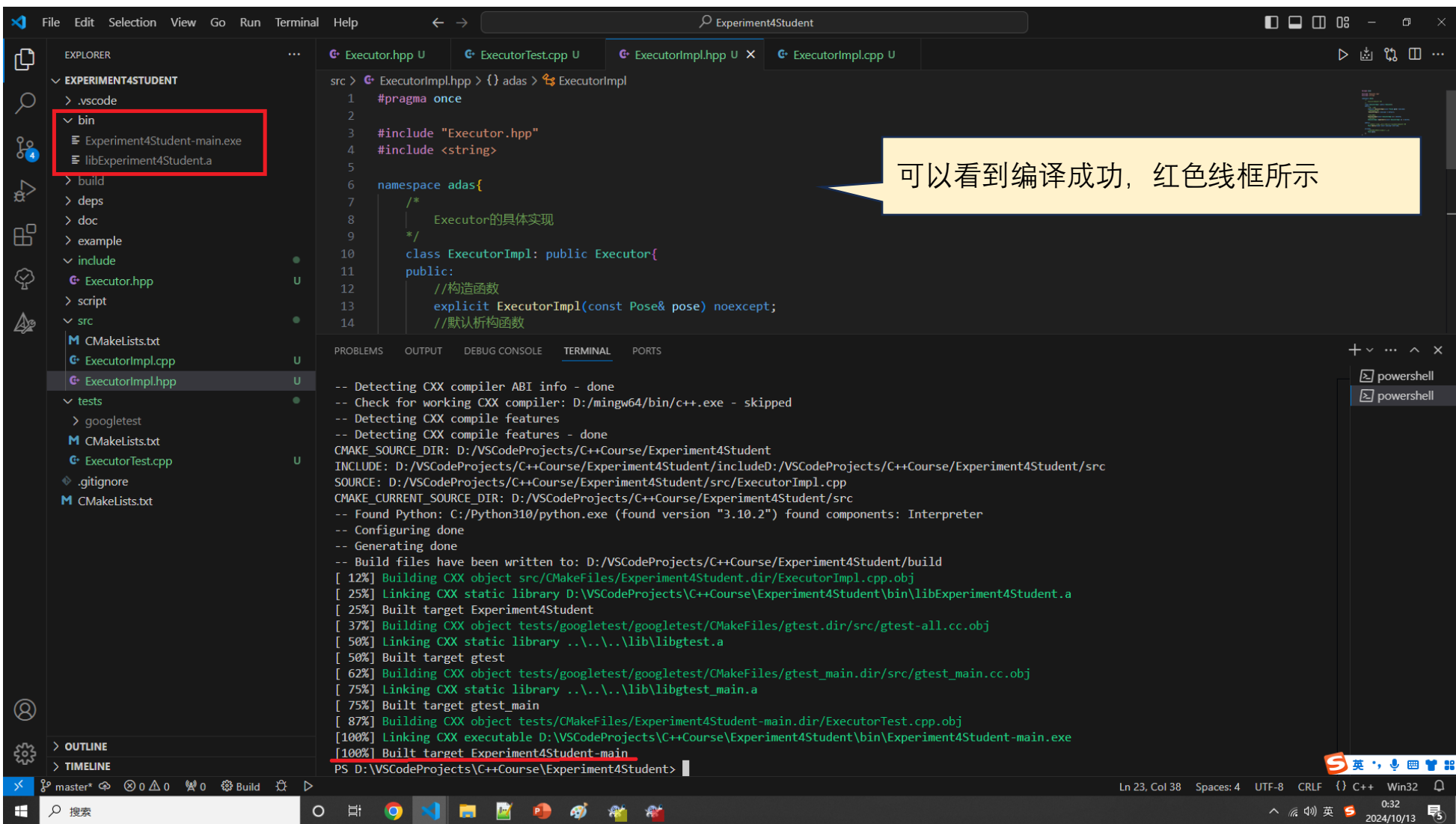
Three callouts provide additional context:

- Callout 1 (top right): "ExecutorImpl构造函数并没有初始化数据成员pose" (The ExecutorImpl constructor does not initialize the data member pose).
- Callout 2 (middle right): "Query方法给出了实现" (The Query method provides the implementation).
- Callout 3 (bottom right): "静态方法NewExecutor方法给出了实现" (The static method NewExecutor provides the implementation).

The status bar at the bottom indicates the current file is 'master*' at line 18, column 67, with 4 spaces, UTF-8 encoding, CRLF line endings, and C++ language. The system tray shows the date and time as 2024/10/13 0:25.

现在来编译工程

打开Terminal，输入下面命令
.\script\build.bat



现在来运行编译好的代码

打开Terminal，输入下面命令

`.\script\run.bat`

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Expected equality of these values:
  target
    Which is: 12-byte object <00-00 00-00 00-00 00-00 45-00 00-00>
  executor->Query()
    Which is: 12-byte object <50-01 AD-00 00-00 00-00 5F-63 6F-6D>
[ FAILED ] ExecutorTest.should_return_init_pose_when_without_command (27 ms)
[ RUN     ] ExecutorTest.should_return_default_pose_when_without_init_and_command
D:\VSCodeProjects\C++Course\Experiment4Student\tests\ExecutorTest.cpp:37: Failure
Expected equality of these values:
  target
    Which is: 12-byte object <00-00 00-00 00-00 00-00 4E-00 00-00>
  executor->Query()
    Which is: 12-byte object <50-01 AD-00 00-00 00-00 5F-63 6F-6D>
[ FAILED ] ExecutorTest.should_return_default_pose_when_without_init_and_command (33 ms)
[-----] 2 tests from ExecutorTest (95 ms total)

[-----] Global test environment tear-down
[=====] 2 tests from 1 test suite ran. (117 ms total)
[ PASSED ] 0 tests.
[ FAILED ] 2 tests, listed below:
[ FAILED ] ExecutorTest.should_return_init_pose_when_without_command
[ FAILED ] ExecutorTest.should_return_default_pose_when_without_init_and_command

2 FAILED TESTS
PS D:\VSCodeProjects\C++Course\Experiment4Student> 
```

可以看到测试失败，因为ExecutorImpl构造函数并没有初始化数据成员pose

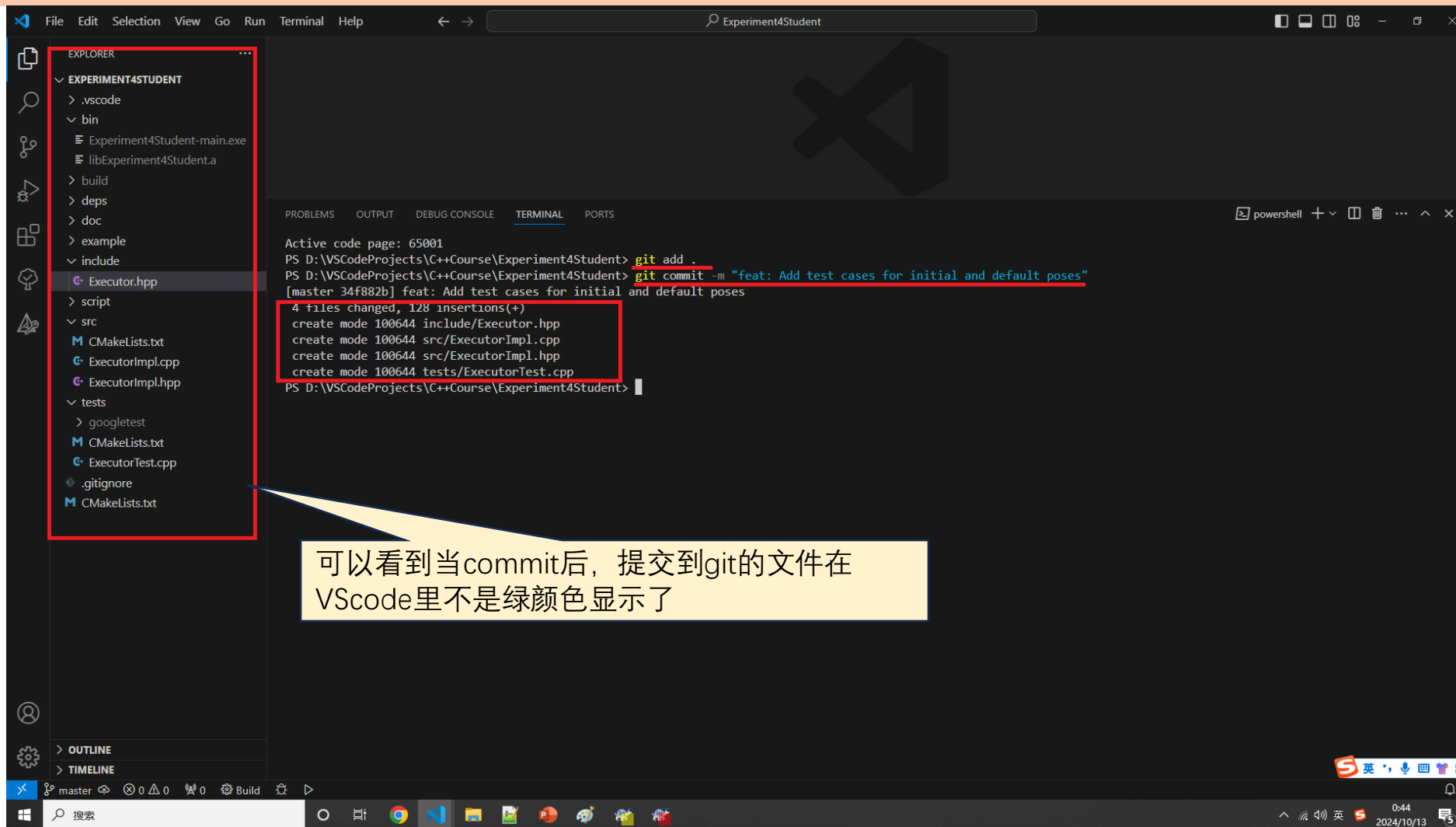
现在将目前的代码提交到Git的本地仓库

打开Terminal，输入下面命令

`git add .`

#提交修改过或新增的文件到暂存区

`git commit -m "feat: Add test cases for initial and default poses"` #将暂存区的文件提交到本地仓库中



通过TortoiseGit查看提交的log日志

D:\VSCodeProjects\C++Course\Experiment4Student - Log Messages - TortoiseGit

master

From: 2024/10/12 To: 2024/10/13

Filter by Subject, Messages, Paths, Authors, Emails, SHA-1, Refname, Notes

Author Email

| Graph | Actions | Message | Author | Date |
|-------|---------|--|--------|---------------------|
| | | Working tree changes | | |
| | | master feat: Add test cases for initial and default poses | guxiwu | 2024/10/13 0:44:30 |
| | | init project | guxiwu | 2024/10/12 22:18:05 |

SHA-1: 34f882b8d3865374c198ecad390db9501a729e2f

* **feat: Add test cases for initial and default poses**

| Path | Extension | Status | Lines added | Lines removed |
|------------------------|-----------|--------|-------------|---------------|
| include/Executor.hpp | .hpp | Added | 37 | 0 |
| src/ExecutorImpl.cpp | .cpp | Added | 22 | 0 |
| src/ExecutorImpl.hpp | .hpp | Added | 30 | 0 |
| tests/ExecutorTest.cpp | .cpp | Added | 39 | 0 |

Showing 2 revision(s), from revision 3c692e29 to revision 34f882b8 - 1 revision(s) selected, 0 file(s) selected; line: 128(+) 0(-) files: modified = 0 added = 4 deleted = 0 replaced = 0

☐ Show Whole Project

☐ All Branches

Filter paths

Refresh

Statistics

Walk Behavior

View

Help

现在修改ExecutorImpl.cpp

在成员初始化列表里初始化pose成员

注意这个文件不是新添加的，而是有了修改，所以颜色变成黄色

```
src > C: ExecutorImpl.cpp > ...
1  #include "ExecutorImpl.hpp"
2
3  #include <new>
4
5  namespace adas{
6      ExecutorImpl::ExecutorImpl(const Pose &pose) noexcept :pose(pose) {}
7
8      Pose ExecutorImpl::Query(void) const noexcept{
9          return pose;
10     }
11
12     /*
13      std::nothrow 是 C++ 标准库中的一个常量，用于指示在分配内存时不抛出任何异常。
14      它是 std::nothrow_t 类型的实例，通常用在 new 运算符和 std::nothrow 命名空间中，
15      以请求内存分配器在分配失败时返回一个空指针，而不是抛出 std::bad_alloc 异常。
16     */
17     Executor *Executor::NewExecutor(const Pose &pose) noexcept{
18         return new(std::nothrow) ExecutorImpl(pose); //只在C++17下有效
19     }
20 }
```

Active code page: 65001
PS D:\VSCodeProjects\C++Course\Experiment4Student> git add .
PS D:\VSCodeProjects\C++Course\Experiment4Student> git commit -m "feat: Add test cases for initial and default poses"
[master 34f882b] feat: Add test cases for initial and default poses
4 files changed, 128 insertions(+)
create mode 100644 include/Executor.hpp
create mode 100644 src/ExecutorImpl.cpp
create mode 100644 src/ExecutorImpl.hpp
create mode 100644 tests/ExecutorTest.cpp
PS D:\VSCodeProjects\C++Course\Experiment4Student>

现在重新来编译和运行工程

打开Terminal，输入下面命令

```
.\script\build.bat
```

```
.\script\run.bat
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> .\script\run.bat
```

```
D:\VSCodeProjects\C++Course\Experiment4Student>rem 自动执行bin目录下的exe文件（如果有多个exe文件，会自动执行最后找到的exe文件）
```

```
D:\VSCodeProjects\C++Course\Experiment4Student\bin\Experiment4Student-main.exe
```

```
Running main() from D:\VSCodeProjects\C++Course\Experiment4Student\tests\googletest\googletest\src\gtest_main.cc
```

```
[=====] Running 2 tests from 1 test suite.
```

```
[-----] Global test environment set-up.
```

```
[-----] 2 tests from ExecutorTest
```

```
[ RUN      ] ExecutorTest.should_return_init_pose_when_without_command
```

```
[          OK ] ExecutorTest.should_return_init_pose_when_without_command (0 ms)
```

```
[ RUN      ] ExecutorTest.should_return_default_pose_when_without_init_and_command
```

```
[          OK ] ExecutorTest.should_return_default_pose_when_without_init_and_command (0 ms)
```

```
[-----] 2 tests from ExecutorTest (31 ms total)
```

```
[-----] Global test environment tear-down
```

```
[=====] 2 tests from 1 test suite ran. (51 ms total)
```

```
[ PASSED ] 2 tests.
```

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> █
```

这是一个里程碑！现在将目前的代码提交到Git的本地仓库

打开Terminal，输入下面命令

git add .

#提交修改过或新增的文件到暂存区

git commit -m "test: Pass initial and default pose test cases" #将暂存区的文件提交到本地仓库中

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Active code page: 65001
PS D:\VSCodeProjects\C++Course\Experiment4Student> git add .
PS D:\VSCodeProjects\C++Course\Experiment4Student> git commit -m "test: Pass initial and default pose test cases"
[master 601abde] test: Pass initial and default pose test cases
 1 file changed, 1 insertion(+), 1 deletion(-)
PS D:\VSCodeProjects\C++Course\Experiment4Student> █
```

Git commit 规范是为了保证团队项目中的提交信息有良好的可读性和可维护性。一条符合规范的commit message应该包含三个部分：Header， Body 和 Footer。

// type和subject必需， scope、 body、 footer可选

<type>(<scope>): <subject>

// 空一行

<body>

// 空一行

<footer>

因此当忽略scope、 body、 footer时， 就是：

<type>: subject

例如我们的二次提交

feat: Add test cases for initial and default poses

test: Pass initial and default pose test cases

小步提交： 每次提交应包含一个小的、独立的功能或修复，便于回溯和管理

清晰的提交信息： 提交信息应简洁明了

| type | 描述 |
|----------|------------------|
| feat | 新增功能 |
| fix | 修复bug |
| UI | 样式调整 |
| refactor | 重构（既不修复错误也不添加功能） |
| perf | 优化相关， 比如提升性能、体验 |
| revert | 回滚之前的commit |

通过TortoiseGit查看提交的log日志

D:\VSCodeProjects\C++Course\Experiment4Student - Log Messages - TortoiseGit

master

From: 2024/10/12 To: 2024/10/13

Filter by Subject, Messages, Paths, Authors, Emails, SHA-1, Refname, Notes

Author Email

| Graph | Actions | Message | Author | Date |
|-------|---------|--|--------|---------------------|
| | | Working tree changes | | |
| | | master test: Pass initial and default pose test cases | guxiwu | 2024/10/13 1:03:38 |
| | | feat: Add test cases for initial and default poses | guxiwu | 2024/10/13 0:44:30 |
| | | init project | guxiwu | 2024/10/12 22:18:05 |

SHA-1: 601abdea42a0e1f3c9be1a5af60d179fddc1edf1

* test: Pass initial and default pose test cases

| Path | Extension | Status | Lines added | Lines removed |
|----------------------|-----------|----------|-------------|---------------|
| src/ExecutorImpl.cpp | .cpp | Modified | 1 | 1 |

Showing 3 revision(s), from revision 3c692e29 to revision 601abdea - 1 revision(s) selected, 0 file(s) selected; line: 1(+) 1(-) files: modified = 1 added = 0 deleted = 0 replaced = 0

Show Whole Project

All Branches

Filter paths

Help

Refresh

Statistics

Walk Behavior

View

Windows taskbar

System tray

现在需要实现执行指令的功能。回顾一下需求

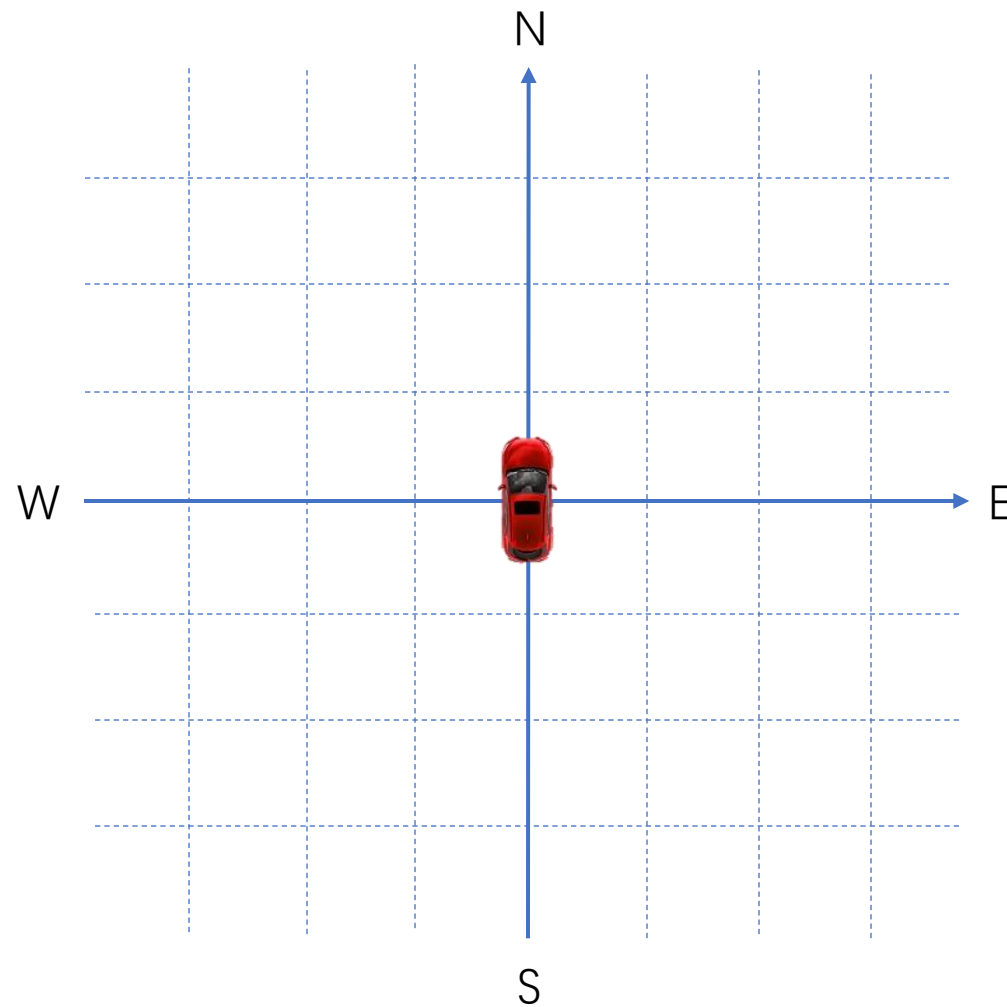
一辆车在二维坐标平面上行驶，车的位置始终为整数表示的坐标点，朝向有4种。

车的控制系统通过以字母表示的指令控制汽车。

指令序列： M L M M R M

当前位置： ~~(0,0)~~ (-2,1)

当前朝向： W



现在需要实现执行指令的功能。回顾一下需求

Executor组件可以执行如下的移动指令：

M: 前进， 1次移动1格

Executor组件可以执行如下的转向指令：

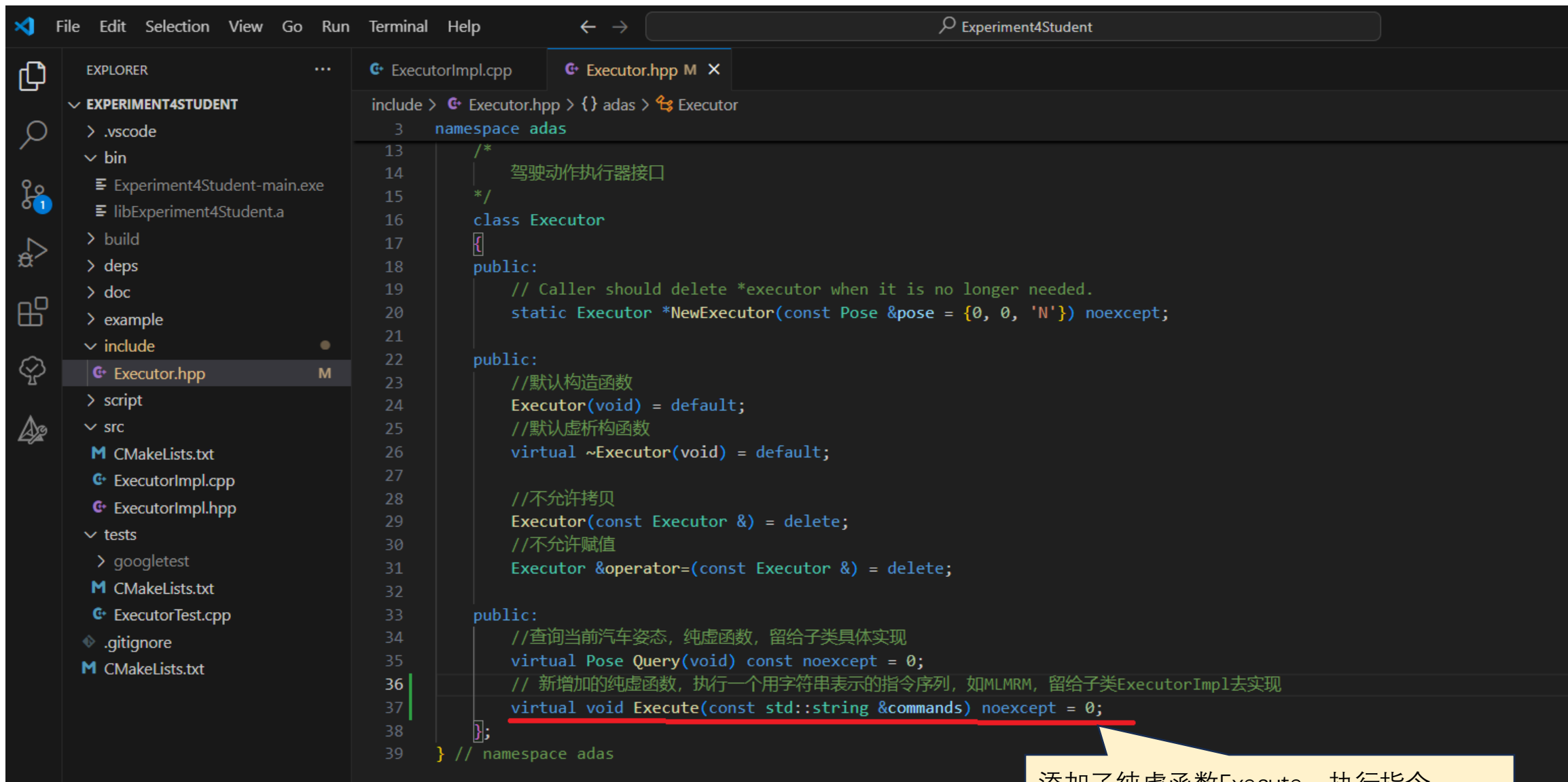
L: 左转90度， 位置不变

R: 右转90度， 位置不变

X轴移动的方向为EW方向， Y轴移动的方向为NS方向。

也可以执行这三个指令的组合序列， 如MLMRM

现在需要在include/Executor.hpp里，为Executor接口添加一个纯虚函数，来执行指令



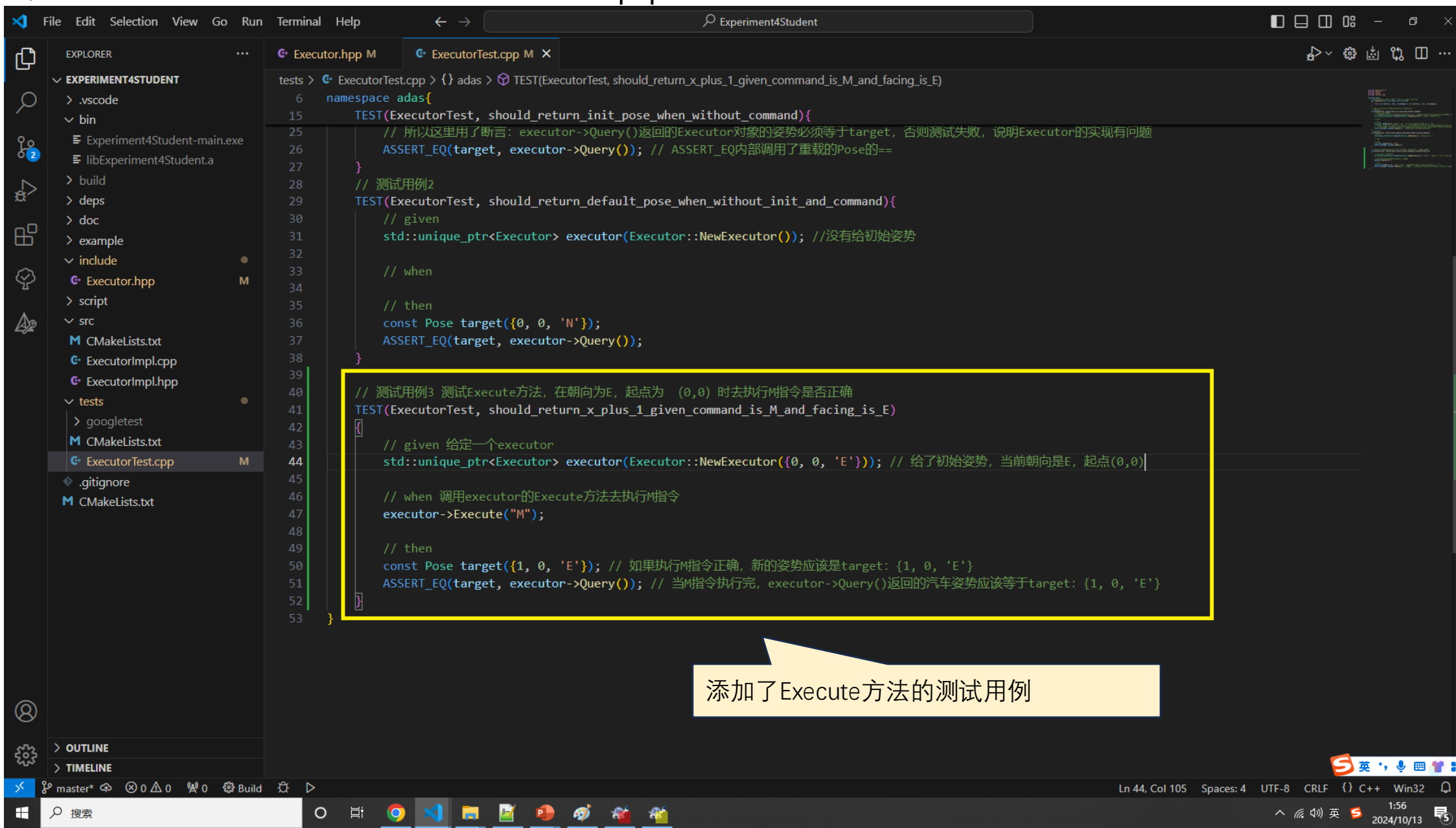
```
File Edit Selection View Go Run Terminal Help
Experiment4Student

EXPLORER
EXPERIMENT4STUDENT
  .vscode
  bin
    Experiment4Student-main.exe
    libExperiment4Student.a
  build
  deps
  doc
  example
  include
    Executor.hpp M
  script
  src
    CMakeLists.txt
    ExecutorImpl.cpp
    ExecutorImpl.hpp
  tests
    googletest
    CMakeLists.txt
    ExecutorTest.cpp
  .gitignore
  CMakeLists.txt

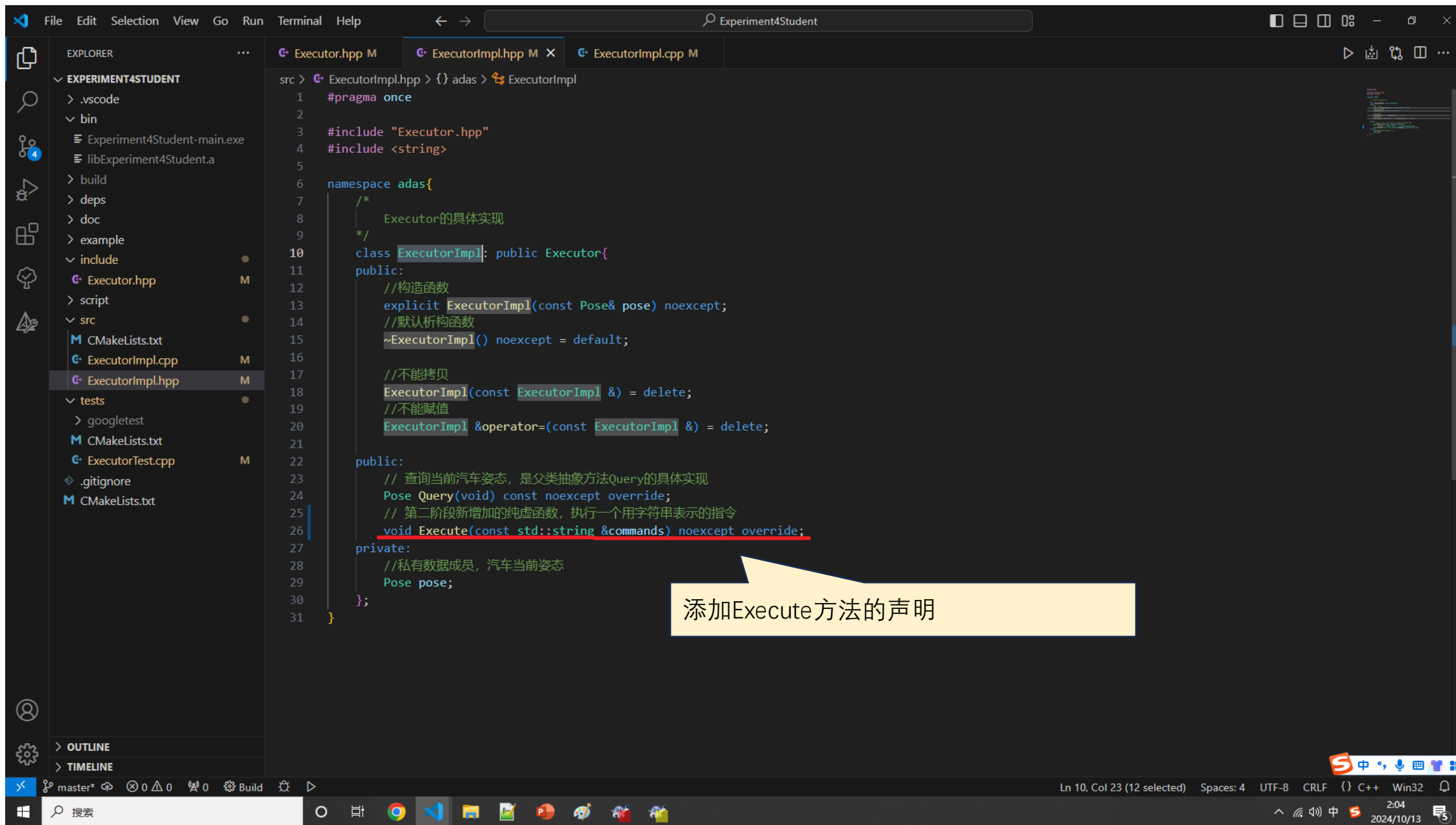
include > Executor.hpp > {} adas > Executor
3 namespace adas
13 /*
14  驾驶动作执行器接口
15 */
16 class Executor
17 {
18 public:
19     // Caller should delete *executor when it is no longer needed.
20     static Executor *NewExecutor(const Pose &pose = {0, 0, 'N'}) noexcept;
21
22 public:
23     //默认构造函数
24     Executor(void) = default;
25     //默认虚析构函数
26     virtual ~Executor(void) = default;
27
28     //不允许拷贝
29     Executor(const Executor &) = delete;
30     //不允许赋值
31     Executor &operator=(const Executor &) = delete;
32
33 public:
34     //查询当前汽车姿态，纯虚函数，留给子类具体实现
35     virtual Pose Query(void) const noexcept = 0;
36     // 新增加的纯虚函数，执行一个用字符串表示的指令序列，如MLMRM，留给子类ExecutorImpl去实现
37     virtual void Execute(const std::string &commands) noexcept = 0;
38 };
39 }
```

添加了纯虚函数Execute，执行指令

修改tests下的ExecutorTest.cpp，添加Execute方法的测试用例



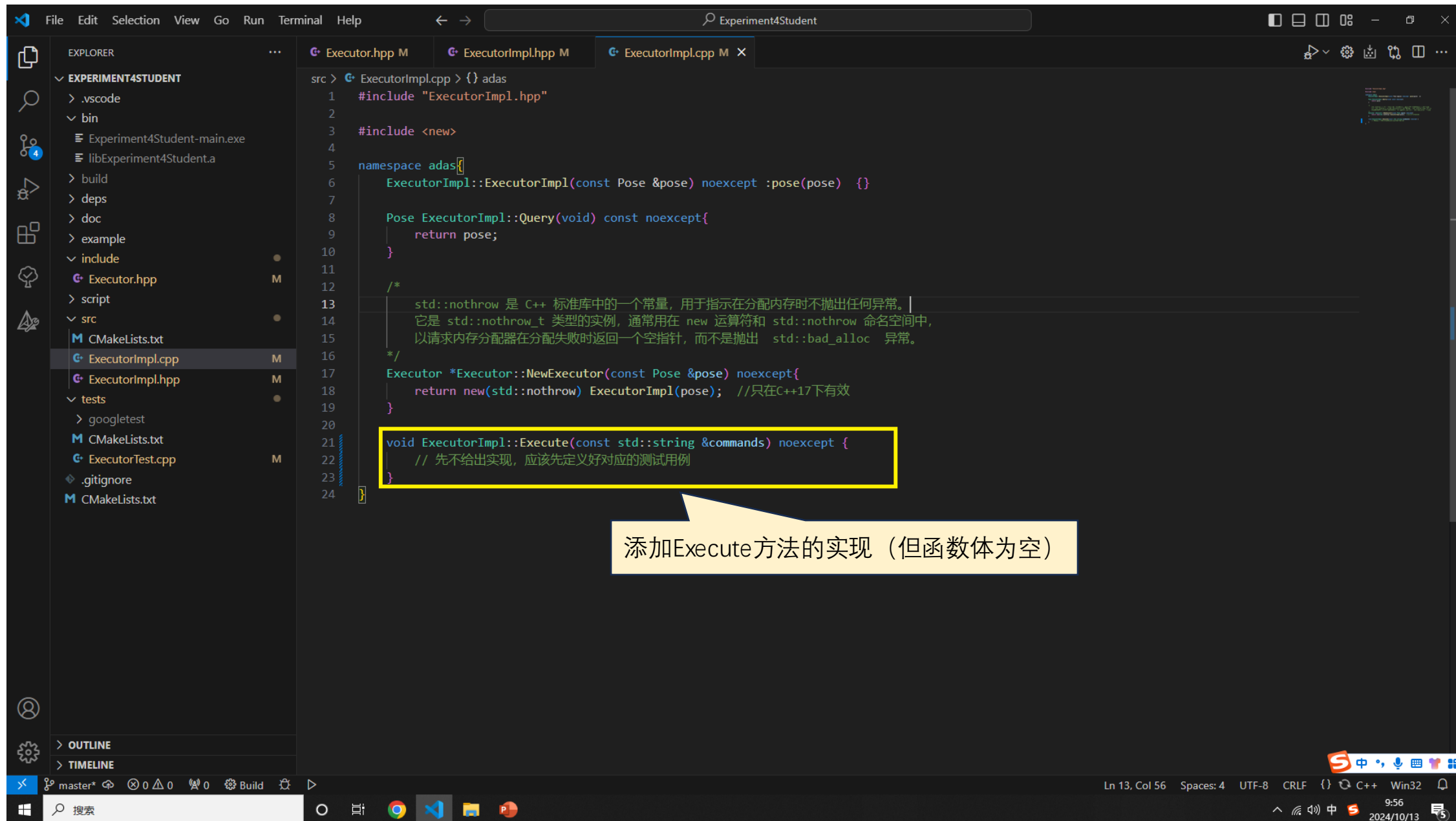
修改src下的ExecutorImpl.hpp，添加Execute方法的声明



```
src > ExecutorImpl.hpp > {} adas > ExecutorImpl
1  #pragma once
2
3  #include "Executor.hpp"
4  #include <string>
5
6  namespace adas{
7      /*
8       * Executor的具体实现
9       */
10     class ExecutorImpl: public Executor{
11     public:
12         //构造函数
13         explicit ExecutorImpl(const Pose& pose) noexcept;
14         //默认析构函数
15         ~ExecutorImpl() noexcept = default;
16
17         //不能拷贝
18         ExecutorImpl(const ExecutorImpl &) = delete;
19         //不能赋值
20         ExecutorImpl &operator=(const ExecutorImpl &) = delete;
21
22     public:
23         // 查询当前汽车姿态，是父类抽象方法Query的具体实现
24         Pose Query(void) const noexcept override;
25         // 第二阶段新增加的纯虚函数，执行一个用字符串表示的指令
26         void Execute(const std::string &commands) noexcept override;
27     private:
28         //私有数据成员，汽车当前姿态
29         Pose pose;
30     };
31 }
```

添加Execute方法的声明

修改src下的ExecutorImpl.cpp，添加Execute方法的实现（但函数体为空）



现在来编译和运行工程

打开Terminal，输入下面命令

```
.\script\build.bat
```

```
.\script\run.bat
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> .\script\build.bat
```

```
D:\VSCodeProjects\C++Course\Experiment4Student>rem 首先配置cmake，然后make编译程序，生成的exe文件放在bin目录下
CMAKE_SOURCE_DIR: D:/VSCodeProjects/C++Course/Experiment4Student
INCLUDE: D:/VSCodeProjects/C++Course/Experiment4Student/includeD:/VSCodeProjects/C++Course/Experiment4Student/src
SOURCE: D:/VSCodeProjects/C++Course/Experiment4Student/src/ExecutorImpl.cpp
CMAKE_CURRENT_SOURCE_DIR: D:/VSCodeProjects/C++Course/Experiment4Student/src
-- Configuring done
-- Generating done
-- Build files have been written to: D:/VSCodeProjects/C++Course/Experiment4Student/build
Consolidate compiler generated dependencies of target Experiment4Student
[ 25%] Built target Experiment4Student
Consolidate compiler generated dependencies of target gtest
[ 50%] Built target gtest
Consolidate compiler generated dependencies of target gtest_main
[ 75%] Built target gtest_main
Consolidate compiler generated dependencies of target Experiment4Student-main
[100%] Built target Experiment4Student-main
PS D:\VSCodeProjects\C++Course\Experiment4Student> █
```

可以看到编译成功，但运行失败

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> .\script\run.bat
```

```
D:\VSCodeProjects\C++Course\Experiment4Student>rem 自动执行bin目录下的exe文件（如果有多个exe文件，会自动执行最后找到的exe文件）
D:\VSCodeProjects\C++Course\Experiment4Student\bin\Experiment4Student-main.exe
Running main() from D:\VSCodeProjects\C++Course\Experiment4Student\tests\googletest\googletest\src\gtest_main.cc
[=====] Running 3 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 3 tests from ExecutorTest
[ RUN      ] ExecutorTest.should_return_init_pose_when_without_command
[ OK       ] ExecutorTest.should_return_init_pose_when_without_command (0 ms)
[ RUN      ] ExecutorTest.should_return_default_pose_when_without_init_and_command
[ OK       ] ExecutorTest.should_return_default_pose_when_without_init_and_command (0 ms)
[ RUN      ] ExecutorTest.should_return_x_plus_1_given_command_is_M_and_facing_is_E
D:\VSCodeProjects\C++Course\Experiment4Student\tests\ExecutorTest.cpp:51: Failure
Expected equality of these values:
  target
    Which is: 12-byte object <01-00 00-00 00-00 00-00 45-00 00-00>
  executor->Query()
    Which is: 12-byte object <00-00 00-00 00-00 00-00 45-00 00-00>
[ FAILED   ] ExecutorTest.should_return_x_plus_1_given_command_is_M_and_facing_is_E (23 ms)
[-----] 3 tests from ExecutorTest (64 ms total)

[-----] Global test environment tear-down
[=====] 3 tests from 1 test suite ran. (84 ms total)
[ PASSED   ] 2 tests.
[ FAILED   ] 1 test, listed below:
[ FAILED   ] ExecutorTest.should_return_x_plus_1_given_command_is_M_and_facing_is_E

1 FAILED TEST
PS D:\VSCodeProjects\C++Course\Experiment4Student> █
```

现在将目前的代码提交到Git的本地仓库

打开Terminal，输入下面命令

git add .

#提交修改过或新增的文件到暂存区

git commit -m "**feat: Add one test cases for move command**" #将暂存区的文件提交到本地仓库中

The screenshot shows the TortoiseGit Log Messages window for the project 'D:\VSCodeProjects\C++\Course\Experiment4Student'. The window displays a list of commits, with the most recent one highlighted: 'feat: Add one test cases for move command' by 'guxiwu' on '2024/10/13 10:12:06'. Below the commit list, the SHA-1 hash '08888e968b1b89ccf32446906dd71e7bdca44dbb' is shown, along with the commit message '* feat: Add one test cases for move command'. A table below the commit message lists the files changed in this commit, with a red box highlighting the table. The table has columns for Path, Extension, Status, Lines added, and Lines removed. The files listed are 'include/Executor.hpp', 'src/ExecutorImpl.cpp', 'src/ExecutorImpl.hpp', and 'tests/ExecutorTest.cpp'. At the bottom of the window, there are buttons for 'Refresh', 'Statistics', 'Walk Behavior', 'View', 'Help', and 'OK'.

| Path | Extension | Status | Lines added | Lines removed |
|------------------------|-----------|----------|-------------|---------------|
| include/Executor.hpp | .hpp | Modified | 2 | 0 |
| src/ExecutorImpl.cpp | .cpp | Modified | 3 | 1 |
| src/ExecutorImpl.hpp | .hpp | Modified | 2 | 1 |
| tests/ExecutorTest.cpp | .cpp | Modified | 14 | 0 |

修改src下的ExecutorImpl.cpp，添加Execute方法的真正实现

我们需要实现的方法是

```
void ExecutorImpl::Execute(const std::string &commands) noexcept { }
```

该方法执行字符串commands给出的指令序列，指令序列是由三个基本指令M、L、R组成的序列

所以需要将commands拆分成一个个的基本指令，然后依次执行每个基本指令

每个基本指令执行完都需要修改汽车的姿势（数据成员pose）

如何将commands拆分成一个个的基本指令，string可以看成char数组，所以如下一个for循环就可以解决

```
for (const auto cmd : commands){  
    //cmd就是当前迭代得到的基本指令  
    //首先需要判断cmd是M? L? R? 在来进行处理  
    //在对每个cmd进行处理时，需要判断汽车当前的姿势情况，进行不同的处理  
}
```

修改src下的ExecutorImpl.cpp，添加Execute方法的真正实现

The image shows a Visual Studio Code editor window with the file Explorer on the left and the Source Explorer on the right. The file Explorer shows the project structure for 'EXPERIMENT4STUDENT', including folders like 'bin', 'build', 'deps', 'doc', 'example', 'include', 'script', and 'src'. The 'src' folder is expanded, showing 'CMakeLists.txt', 'ExecutorImpl.cpp', and 'ExecutorImpl.hpp'. The Source Explorer shows the 'ExecutorImpl.cpp' file, which is open in the editor. The code in the editor is as follows:

```
src > ExecutorImpl.cpp > {} adas
1  #include "ExecutorImpl.hpp"
2
3  #include <new>
4
5  namespace adas{
6      ExecutorImpl::ExecutorImpl(const Pose &pose) noexcept :pose(pose) {}
7
8      Pose ExecutorImpl::Query(void) const noexcept{ ...
9
10     /* ...
11
12     Executor *Executor::NewExecutor(const Pose &pose) noexcept{ ...
13
14     /*
15     执行字符串commands给出的指令序列，指令序列是由三个基本指令M、L、R组成的序列
16     所以需要将commands拆分成一个个的基本指令，然后依次执行每个基本指令
17     每个基本指令执行完都需要修改汽车的姿势（数据成员pose）
18     如何将commands拆分成一个个的基本指令，string可以看成char数组，所以如下一个for循环就可以解决
19     for (const auto cmd : commands){
20         //cmd就是当前迭代得到的基本指令
21         //首先需要判断cmd是M? L? R? 在来进行处理
22         //在对每个cmd进行处理时，有需要判断汽车当前的姿势情况，进行不同的处理
23     }
24
25     */
26
27     void ExecutorImpl::Execute(const std::string &commands) noexcept {
28         //遍历commands里面的每个指令cmd
29         for (const auto cmd : commands)
30         {
31             //如果是M指令
32             if (cmd == 'M')
33             { // 如果是M指令，则需要根据当前汽车姿势的heading（朝向）决定如何移动车辆（重新计算坐标）
34                 if (pose.heading == 'E') { ++pose.x; }
35                 else if (pose.heading == 'W') { --pose.x; }
36                 else if (pose.heading == 'N') { ++pose.y; }
37                 else if (pose.heading == 'S') { --pose.y; }
38             }
39         }
40     }
41 }
```

A yellow box highlights the code for handling 'M' commands, and a callout bubble points to it with the text "实现了M指令的执行".

现在来编译和运行工程

打开Terminal，输入下面命令

```
.\script\build.bat
```

```
.\script\run.bat
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> .\script\build.bat
```

```
D:\VSCodeProjects\C++Course\Experiment4Student>rem 首先配置cmake, 然后make编译程序, 生成的exe文件放在bin目录下
CMAKE_SOURCE_DIR: D:/VSCodeProjects/C++Course/Experiment4Student
INCLUDE: D:/VSCodeProjects/C++Course/Experiment4Student/includeD:/VSCodeProjects/C++Course/Experiment4Student/src
SOURCE: D:/VSCodeProjects/C++Course/Experiment4Student/src/ExecutorImpl.cpp
CMAKE_CURRENT_SOURCE_DIR: D:/VSCodeProjects/C++Course/Experiment4Student/src
-- Configuring done
-- Generating done
-- Build files have been written to: D:/VSCodeProjects/C++Course/Experiment4Student/build
Consolidate compiler generated dependencies of target Experiment4Student
[ 12%] Building CXX object src/CMakeFiles/Experiment4Student.dir/ExecutorImpl.cpp.obj
[ 25%] Linking CXX static library D:\VSCodeProjects\C++Course\Experiment4Student\bin\libf
[ 25%] Built target Experiment4Student
Consolidate compiler generated dependencies of target gtest
[ 50%] Built target gtest
Consolidate compiler generated dependencies of target gtest_main
[ 75%] Built target gtest_main
Consolidate compiler generated dependencies of target Experiment4Student-main
[ 87%] Linking CXX executable D:\VSCodeProjects\C++Course\Experiment4Student\bin\Experi
[100%] Built target Experiment4Student-main
PS D:\VSCodeProjects\C++Course\Experiment4Student> █
```

可以三个测试都通过

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Active code page: 65001

```
PS D:\VSCodeProjects\C++Course\Experiment4Student> .\script\run.bat
```

```
D:\VSCodeProjects\C++Course\Experiment4Student>rem 自动执行bin目录下的exe文件 (如果有多个exe文件, 会自动执行最后找到的exe文件)
D:\VSCodeProjects\C++Course\Experiment4Student\bin\Experiment4Student-main.exe
Running main() from D:\VSCodeProjects\C++Course\Experiment4Student\tests\googletest\googletest\src\gtest_main.cc
[=====] Running 3 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 3 tests from ExecutorTest
[ RUN      ] ExecutorTest.should_return_init_pose_when_without_command
[ OK       ] ExecutorTest.should_return_init_pose_when_without_command (0 ms)
[ RUN      ] ExecutorTest.should_return_default_pose_when_without_init_and_command
[ OK       ] ExecutorTest.should_return_default_pose_when_without_init_and_command (0 ms)
[ RUN      ] ExecutorTest.should_return_x_plus_1_given_command_is_M_and_facing_is_E
[ OK       ] ExecutorTest.should_return_x_plus_1_given_command_is_M_and_facing_is_E (0 ms)
[-----] 3 tests from ExecutorTest (43 ms total)

[-----] Global test environment tear-down
[=====] 3 tests from 1 test suite ran. (64 ms total)
[ PASSED  ] 3 tests.
PS D:\VSCodeProjects\C++Course\Experiment4Student> █
```

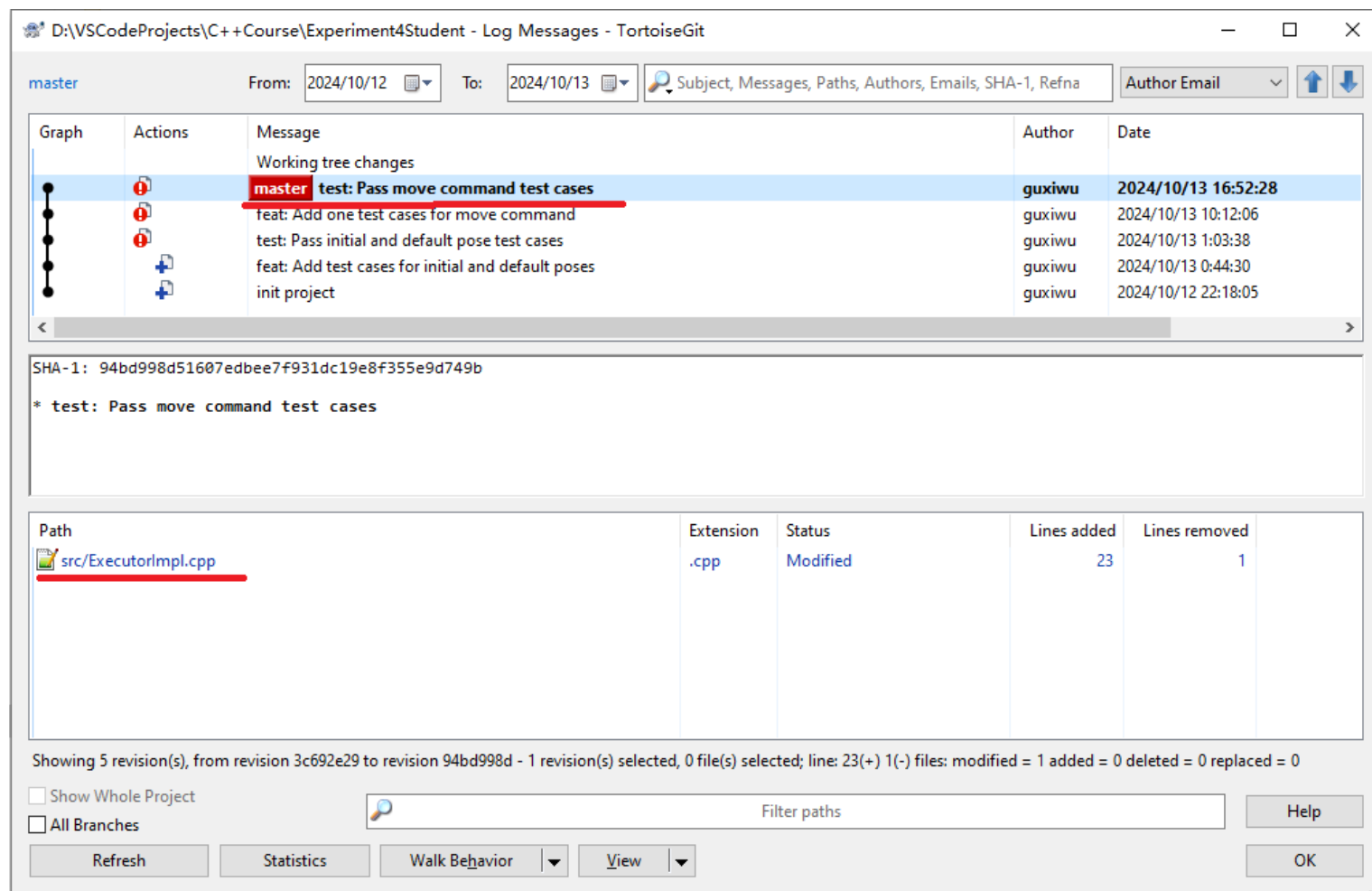
现在将目前的代码提交到Git的本地仓库

打开Terminal，输入下面命令

```
git add .
```

```
git commit -m "test: Pass move command test cases" #提交修改过或新增的文件到暂存区
```

#将暂存区的文件提交到本地仓库中



剩下需要自己完成的工作

| 方向 | E | W | N | S |
|----|---------------------|---------------------|---------------------|---------------------|
| M | 当前朝向E 执行M X+1 | 当前朝向W 执行M X-1 | 当前朝向N 执行M Y+1 | 当前朝向S 执行M Y-1 |
| L | 当前朝向E 执行L 朝向N | 当前朝向W 执行L 朝向S | 当前朝向N 执行L 朝向W | 当前朝向S 执行L 朝向E |
| R | 当前朝向E 执行R 朝向S | 当前朝向W 执行R 朝向N | 当前朝向N 执行R 朝向E | 当前朝向S 执行R 朝向W |

三个指令，每个指令四种情况，共12个测试
目前已经完成了M指令的一个测试用例及实现了M指令的执行

请完成M指令另外三个测试用例
请完成L指令四个测试用例，以及实现L指令的执行
请完成R指令四个测试用例，以及实现L指令的执行

每个测试用例的test_name这里都已经给出

- 移动指令
- should_return_x_plus_1_given_command_is_M_and_facing_is_E
 - should_return_x_minus_1_given_command_is_M_and_facing_is_W
 - should_return_y_plus_1_given_command_is_M_and_facing_is_N
 - should_return_y_minus_1_given_command_is_M_and_facing_is_S
- 左转指令
- should_return_facing_N_given_command_is_L_and_facing_is_E
 - should_return_facing_W_given_command_is_L_and_facing_is_N
 - should_return_facing_S_given_command_is_L_and_facing_is_W
 - should_return_facing_E_given_command_is_L_and_facing_is_S
- 右转指令
- should_return_facing_S_given_command_is_R_and_facing_is_E
 - should_return_facing_W_given_command_is_R_and_facing_is_S
 - should_return_facing_N_given_command_is_R_and_facing_is_W
 - should_return_facing_E_given_command_is_R_and_facing_is_N

剩下工作的git提交

剩下工作需要进行如下提交

1) 完成了M指令的另外三个测试用例，运行成功以后

git add .

git commit -m "test: Add and Pass all move command test cases"

2)完成L指令四个测试用例，但还没有实现L指令的执行。这时编译成功，测试失败

git add .

git commit -m "feat: Add test cases for turn left command"

3) 实现L指令的执行， L指令四个测试用例测试通过

git add .

git commit -m "test: Pass turn left command test cases"

4) 完成R指令四个测试用例，但还没有实现R指令的执行。这时编译成功，测试失败

git add .

git commit -m "feat: Add test cases for turn right command"

5) 实现R指令的执行， L指令四个测试用例测试通过

git add .

git commit -m " test: Pass turn right command test cases"

实验检查

因为每个同学的实验工程下都有git目录，因此每次实验完成后，需要提交工程（打包）到学习平台。老师和助教会解压每个同学的工程，首先会测试运行所有的测试用例，其次会检查代码的编写质量，最后会用TortoiseGit去检查每个同学的提交日志，看是否符合要求

实验报告和实验评测

由于四次实验是逐次递进，所以最后实验完成后，针对最终完成的实验内容撰写实验报告。

实验分数最终由：实验考勤(10%)+实验完成情况（60%）+实验报告（30%）组成

其中实验完成情况的分数：根据检查每次实验提交的工程（包括运行测试程序，检查代码质量，检查代码提交日志）的情况来评定。每次实验都会检查一次，四次实验完成情况的平均分作为实验完成情况的分数。