Seed 选择模式现在设置为 "Off"，你的知识库将不会被使用。

```cpp
    system("title Lightweight Task Scheduler - Project 3");

    cout << "Initializing Scheduler..." << endl;

    // 初始化UI并运行主循环
    ConsoleUI ui;
    ui.ShowMenu();

    cout << "Stopping Scheduler..." << endl;
    TaskScheduler::Instance().Stop();

    cout << "Goodbye!" << endl;
    return 0;
}
```

代码功能说明

该代码严格遵循了图中的设计要求：

1. **ITask 接口**：定义在第 75 行，包含 `GetName` 和 `Execute` 。
2. **具体任务 (A-E)**：
   a. **Task A (文件备份)**：第 85 行，模拟文件写入 `backup_data.zip` 。
   b. **Task B (矩阵乘法)**：第 105 行，计算 200x200 矩阵并计时。
   c. **Task C (HTTP GET)**：第 132 行，调用系统 `curl` 命令并将结果存入 `zen.txt` 。
   d. **Task D (课堂提醒)**：第 147 行，使用 `MessageBoxA` 弹出窗口（运行在分离线程中以防阻塞调度器）。
   e. **Task E (随机数统计)**：第 160 行，生成1000个随机数并计算均值/方差。
3. **TaskScheduler (调度器)**：
   a. **Singleton**：第 276 行 `Instance()` 方法。
   b. **Priority Queue**：第 208 行，存储 `ScheduledTask` ，按执行时间排序。
   c. **Worker Thread**：第 229 行 `WorkerLoop` ，负责取出任务并执行。
   d. **Mutex/Condition Variable**：用于线程安全的任务添加和等待。
4. **LogWriter (日志)**：第 29 行，RAII封装，程序启动打开，退出关闭，所有任务执行都会写入 `scheduler.log` 。
5. **设计模式**：
   a. **Factory**：第 327 行 `TaskFactory` 。
   b. **Observer**：UI 类 (`ConsoleUI`) 继承自 `IObserver` ，当任务状态变化时，调度器调用 `NotifyObservers` ，UI实时打印通知。

如何运行

1. **环境**：Windows系统。
2. **编译器**：Visual Studio (MSVC) 或 MinGW (g++)。
3. **编译命令 (MinGW)**：

```bash
g++ main.cpp -o scheduler.exe -std=c++17 -static
```

4. **操作**：
   a. 运行程序。
   b. 输入 `A` 、 `B` 、 `C` 、 `D` 或 `E` 并回车添加任务。
   c. 观察控制台输出（UI更新）以及生成的 `scheduler.log` 文件。
   d. 对于 Task D，会弹出一个 Windows 对话框。

Gemini 3 Pro Preview 12-10 14:34

代码报错如图所示，认真修复，保证代码能运行

我 12-10 14:39

继续优化TaskE生成的数字矩阵右边竖线离数字太远的问题

我 12-10 16:25

升级计划

您已用完所有额度，升级以保持工作流程顺畅。

0 额度

Process >

是 **Visual Studio 默认使用 C++14 标准**，而 `<filesystem>` 库需要 **C++17** 标准才能支持。

个问题，我修改了代码，**移除了对 C++17 `<filesystem>` 的依赖**，改用标准的文件流操作。这样即，代码也能直接在 Visual Studio 中编译运行。

了 `#define NOMINMAX` 以防止 Windows 头文件与 C++ 标准库冲突。

下修复后的代码覆盖你原来的文件：

```cpp
_SECURE_NO_WARNINGS // 忽略 unsafe 警告
NMAX               // 防止 Windows 宏定义 min/max 冲突

stream>
ring>
ctor>
eue>
tex>
ndition_variable>
read>
nctional>
stream> // 文件流
mory>
rono>
ime>
manip>
stream>
ndom>
p>
ndows.h> // 用于Task D弹窗和Sleep

ace std;

==================================
块 (RAII & 单例辅助)
==================================
ter {

 logFile;
gMutex;
r() {
使用追加模式打开日志
ile.open("scheduler.log", ios::app);
(logFile.is_open()) {
 logFile << "--- Log System Started ---\n";



er() {
(logFile.is_open()) {
 logFile << "--- Log System Closed ---\n";
 logFile.close();



s Singleton
ogWriter& Instance() {
ic LogWriter instance;
rn instance;


te(const string& msg) {
_guard<mutex> lock(logMutex);
 now = chrono::system_clock::to_time_t(chrono::system_clock::now());
(logFile.is_open()) {
 // 使用更兼容的时间格式化方式
 struct tm timeInfo;
 localtime_s(&timeInfo, &now);
 logFile << "[" << put_time(&timeInfo, "%Y-%m-%d %H:%M:%S") << "] " << msg << endl;
 logFile.flush(); // 确保写入




==================================
模式 (Observer Interface)
==================================
ver {

 void OnTaskEvent(const string& message) = 0;
 ~IObserver() = default;



==================================
口与策略 (Strategy Pattern)
==================================
 {

 string GetName() const = 0;
 void Execute() = 0;
 ~ITask() = default;

文件备份 (模拟压缩)
```

开启新的话题