1.随机数

简介

编写一个程序,随机产生10个$[1,100]$的随机正整数,利用$vector$容器存储数据.

要求

使用$std::accumulate$算法求者10个随机数之和,$std::sort$算法对他们进行排序,用$std::copy$算法输出排序后的这10个随机数.

2.银行账户

简介

声明一个银行账户类$Account$,该类有账号id,余额$balence$两个数据成员，有获取账号，获取余额，存款和取款的函数，以及必要的构造函数。

要求

请按上述要求声明该银行账户类并在$main$函数中定义该类的多个对象，然后对它们进行存取款和查询余额的操作。

3.Building

简介

设计一个基类$Building$类 有楼房的层数，房间数和总面积，再由基类派生出教学楼类$TeachBuildin$g类和宿舍楼类$DormBuilding$类，教学楼类增加教室数，宿舍楼类 增加宿舍数，容纳学生总人数。要求对每个类的成员变量实现增删改查等操作。

要求

基类和派生类的成员函数根据自己需要提供.

4.Base

简介

设计一个基类$Base$为抽象类，其中包含$setTitle()$和$showTitile()$两个成员函数，另有一个纯虚函数$isGood()$. 由此派生图书类$Book$和杂志类$Journal$,分别实现纯虚函数$isGood()$ 对于前者,如果每月图书销售量大于500,则返回$true$;对于后者,如果每月杂志销售量超过2500,则返回$true$.

要求

设计这三个类并在$main$函数中测试.

5. 复数运算

简介

自定义一个复数类$Complex$,重载运算符$+$和$-$,使其能够直接用于复数的加减法

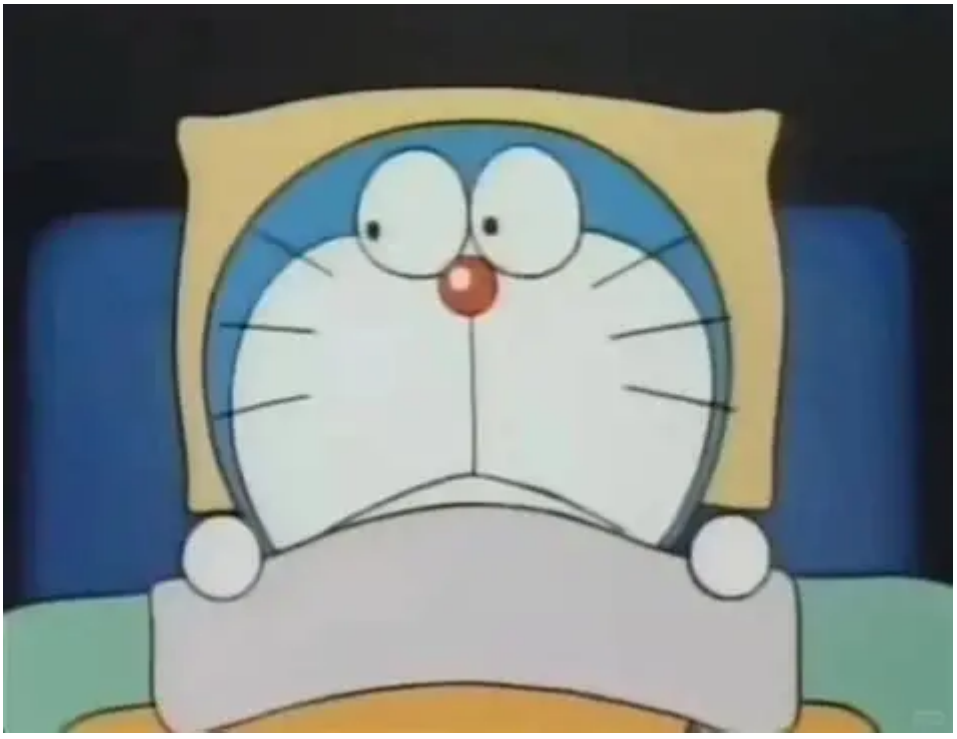
要求

- 将运算符重载为$Complex$类的成员函数
- 重载流插入运算符$<<$和流提取运算符$>>$,使其能用于复数的输入输出

扩展

要求添加功能实现一个复数和一个$double$类型实数的加减运算(结果还是一个复数),其他功能不变.

6.睡觉排序zzz



大家常用的排序算法有 快速排序、归并排序、基数排序等,
哆啦A梦不想写这些排序，它想出了一个颠覆传统排序方法的算法, 没错, 就是睡觉排序!
睡觉排序的原理就是同时启动多个线程, 让它们睡觉, 各个线程睡觉的时间是待排序数组中的一个值乘以共同的缩放倍数,
先汇合的线程夺锁并把自己持有的值加入答案中, 等待所有线程汇合, 排序就完成了

题目要求:

1. 读懂已给出的代码, 无需修改
2. 完成sleepSortAssist函数, 把它持有的值加入ans

3. 完成launchSyncThread函数, 利用latch同步启动多个sleepSortAssist

提示:

1. 修改多个线程共享的资源时注意上锁
2. 注意各个线程睡觉时间的缩放

样例输出:

```
Before Sort: 10 1 3 6 7 6 5 1 9 9 2 1 6 8 2 4
After  Sort: 1 1 1 2 2 3 4 5 6 6 6 7 8 9 9 10
```

参考代码

```
#include <iostream>
#include <vector>
#include <utility>
#include <algorithm>
#include <random>
#include <functional>
#include <mutex>
#include <shared_mutex>
#include <condition_variable>
#include <chrono>
#include <thread>
#include <latch>
namespace stupid_algo {
class SleepSort {
protected:
    /* 原始数据 */
    std::vector<int> data_;
    /* 线程池 */
    std::vector<std::thread> pool_;
    /* 需要用到的锁 */
    std::mutex ans_mtx_;
    /* 答案保存到ans中 */
    std::vector<int> ans_;
    /* 完成sleepSortAssist函数, 把它持有的值加入ans_ */
    void sleepSortAssist(int val) {


        /* Code Here*/



    }
```

```
public:
    template<typename VectorInt>
    explicit SleepSort(VectorInt&& data) : data_(std::forward<VectorInt>
(data)) {}
    std::vector<int>& getAns() { return ans_; }
    /* 完成launchSyncThread函数，利用std::latch让多个sleepSortAssist线程同时启动
*/
    void launchSyncThread() {
        const int n = static_cast<int>(data_.size());
        std::latch sync_point(n);
        // Ensure that sleep-thread launch at the same time.
        std::function<void(int)> task = [&](const int val) -> void {



            /* Code Here*/



        };
        for(int i=0; i<n; ++i) pool_.emplace_back(task, data_[i]);
        for(int i=0; i<n; ++i) pool_[i].join();
    }
};
}
int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    std::vector<int> vec{1,1,1,2,2,3,4,5,6,6,6,7,8,9,9,10};
    std::mt19937 mt(std::random_device{}());
    std::ranges::shuffle(vec, mt);
    std::cout<<"Before Sort: ";
    for(const auto& i : vec) std::cout<<i<<" ";
    std::cout<<std::endl;
    stupid_algo::SleepSort sb(std::move(vec));  // SB = " SleepSort Behavior
"
    sb.launchSyncThread();
    std::cout<<"After  Sort: ";
    for(const auto& i : sb.getAns()) std::cout<<i<<" ";
    std::cout<<std::endl;
    return 0;

}
```

7. 迷宫寻路_双人版

非常经典的BFS模板题目，但是多线程，嘻嘻 :D
学习原子变量的用法，能够分辨原子和锁的不同使用场景

前置算法知识

1. 学习广度优先搜索 即BFS
   如果你不知道什么叫BFS，可以看看下面这些资料
   https://oi-wiki.org/search/bfs/
   https://www.bilibili.com/video/BV1vKQaYSEm3

前置STL知识

1. 学习c++11的 std::atomic
2. 学习c++20的 std::stop_token　std::stop_source

注意事项

1. 使用c++20以上的标准编译

题目描述

jyw的二刺猿老婆被困在一个矩形迷宫里。
迷宫可以视为一个 n × m 矩阵，每个位置要么是空地，要么是墙
迷宫中的人只能从一个空地走到其上、下、左、右的空地。
jyw初始位于迷宫左上角(0,0)的位置，
jyw的二刺猿老婆初始位于迷宫右下角(n-1,m-1)的位置。

现在jyw想知道能否在这个迷宫中见到对方。
于是他打电话和ShiinaHiyori约定，两人各自启动一个线程，一起在迷宫中寻找对方。
为了更快地见到对方，两人都会在自己走过的空地上留下自己的标记。
如果两人见到了对方的标记，就说明两人可以见面，真是太好了。
此时在控制台输出Yes
如果两人标记了自己能到达的所有空地，仍然无法看到对方的标记，就说明两人无法见到。
此时在控制台输出No

题目要求

可以参考我给出的挖空的多线程bfs模板，补全代码，也可以自己从头编写。
只要最终实现两个线程共同完成bfs的效果即可。
你甚至可以直接复制这道题目的原题的题解，但是你要把它修改成多线程版本。

提示

1. 注意使用exchange而不是先load再store，防止错过对方标记
2. 注意线程退出机制，可以用到std::stop_source

输入格式

第一行，两个正整数 n,m。
接下来 n 行，输入这个迷宫。每行输入一个长为 m 的字符串，# 表示墙，. 表示空地。

输出格式

仅一行，一个字符串。如果两人能见到，则输出 Yes；否则输出 No

样例输入1:

```
3 5
.##.#
.#...
...#.
```

样例输出1:

```
Yes
```

样例输入2:

```
4 5
....#
.###.
#..#.
```

```
.....
```

样例输出2:

```
No
```

参考代码

```
#include <algorithm>
#include <vector>
#include <queue>
#include <cstdint>
#include <utility>
#include <functional>
#include <thread>
#include <array>
#include <stop_token>
#include <cstdio>
class BFS_2D_MultiThread {
public:

    struct AtomicInt {
        std::atomic<int> data;

        AtomicInt() { data.store(0); }
        AtomicInt(int x) { data.store(x); }

        AtomicInt(const AtomicInt&) = delete;
        AtomicInt& operator=(const AtomicInt&) = delete;

        AtomicInt(AtomicInt&& other) noexcept : data(other.data.load()) {}
        AtomicInt& operator=(AtomicInt&& other) noexcept {
            data.store(other.data.load());
            return *this;
        }    };
private:
    static constexpr std::array<std::array<int,2>,4> direction = { { {1,0},
{-1,0}, {0,1}, {0,-1} } };

    std::pair<int,int> start_;
    std::pair<int,int> end_;
    std::vector<std::vector<int>> matrix_;
```

```
    int edge_x;
    int edge_y;

    std::vector<std::vector<AtomicInt>> visited_;

    std::stop_source sts_;

    void bfs_assist_(const std::pair<int,int> s, const int id) {



        /* your code */

    }

    std::thread bfs_thread_one;
    std::thread bfs_thread_two;

    bool is_bfs_finish = false;

    std::atomic<bool> ans_;

public:
    BFS_2D_MultiThread() = delete;

    template <typename Graph>
    BFS_2D_MultiThread(std::pair<int,int> start, std::pair<int,int> end,
Graph&& graph) :
        start_(start),
        end_(end),
        matrix_(std::forward<Graph>(graph)),
        edge_x(static_cast<int>(matrix_.size())),
        edge_y(static_cast<int>(matrix_[0].size()))
    {       ans_.store(false);

        visited_.resize(edge_x);
        for(auto& raw : visited_) raw.resize(edge_y);

        visited_[start.first][start.second].data.exchange(1);
        visited_[end.first][end.second].data.exchange(2);
    }
    void launch_bfs() {
        if(is_bfs_finish) return;
        this->bfs_thread_one =
std::thread(std::bind(&BFS_2D_MultiThread::bfs_assist_,this,start_,1));
        this->bfs_thread_two =
std::thread(std::bind(&BFS_2D_MultiThread::bfs_assist_,this,end_,2));
        bfs_thread_one.join();
        bfs_thread_two.join();
        is_bfs_finish = true;
```

```
    }
    bool is_connected() { return ans_.load(); }

};

template <typename UINT>
UINT read(){
    UINT x = 0;
    char c =  getchar();
    while( c<'0' || c>'9' ) { c = getchar(); }
    while( '0'<=c && c<='9' ) { x = x*10 + c - '0'; c = getchar(); }
    return x;
}

template <typename UINT>
void write(UINT x){
    if(x<10) { putchar(x+'0'); return;}
    write(x/10);
    putchar(x%10 + '0');
}

static inline void solve() {
    int n = read<int>();
    int m = read<int>();
    std::vector<std::vector<int>> grid_graph(n,std::vector<int>(m,0));
    for(int i=0; i<n; ++i) {
        for(int j=0; j<m; ++j) {
            grid_graph[i][j] = (getchar()=='#');
        }        getchar();
    }    BFS_2D_MultiThread bfs2d(std::pair<int,int>{0,0},
std::pair<int,int>{n-1,m-1}, std::move(grid_graph));
    bfs2d.launch_bfs();
    if(bfs2d.is_connected()) puts("Yes");
    else puts("No");
}

int main() {
    solve();
    return 0;
}
```

提交注意事项

- 将七道题的源代码放置在一个以班级姓名命名的压缩包内
- 提交给群内管理员学长 @牢大 @牢弟
- 视完成情况可能会有线下验收

- 提交期限为开始做题的十天内