OOP 大作业-Matrix

负责助教：冯思远
场外助教组：任云玮

目录

1 概要

这是课程 [程序设计 2017] 的 OOP 大作业，请同学们按照文档中的要求，完成文件 `Matrix.hpp` 中的内容并回答相关问题（粗体内容）。

2 实现要求

1. A 班：手动维护一维数据结构，以行优先方式实现二维表示，禁止使用 vector、deque 等**任何 STL 容器**，需通过正确性测试和鲁棒性测试。
2. B 班：无任何限制，仅需通过正确性测试。
3. **强烈建议但不强制** B 班有基础同学完成所有或部分 A 班要求。

3 接口

3.1 构造函数与赋值

1. 默认构造函数；
2. `Matrix(std::size_t n, std::size_t m, T init = T())`，构造一个大小为 $n \times m$ 的矩阵，并将里面的每个元素初始化为 `init`
3. `Matrix(std::initializer_list<std::initializer_list<T>>)`，利用给定的 `initializer_list` 来构造一个矩阵。
4. “拷贝”构造函数与赋值。要求如果可以用一个 `T` 来构造一个 `V`，则可以使用一个 `Matrix<T>` 来构造一个 `Matrix<V>`。
5. （仅 A 班）“移动”构造函数与赋值。仅要求相同类型的移动构造。**思考：为什么不能跨类型移动**

3.2 元素获取

完成如下两个函数（请务必写对），它们用于获取矩阵的第 i 行 j 列的元素，0-based。并说明**为何需要下述两个重载**

1. `T& operator()(std::size_t i, std::size_t j)`
2. `const T& operator()(std::size_t, std::size_t) const`

并完成如下两个函数，返回矩阵的第 i 行／列。

1. `Matrix<T> row(std::size_t i) const`
2. `Matrix<T> column(std::size_t i) const`

3.3 运算

3.3.1 一元运算

1. `-`，对矩阵中每个元素取负。
2. `-=`
3. `+=`
4. `*=`，数乘
5. `Matrix tran() const`，返回当前矩阵的转置矩阵。注：不改变当前矩阵。

要求对于 `+=` 等运算，支持 `Matrix<int>+=` 一个 `Matrix<double>`

3.3.2 二元运算

1. `==`，比较两个矩阵是否相同（元素的值相等即可）。
2. `!=`，比较两个矩阵是否不同。
3. `+`，定义为友元函数
4. `-`，定义为友元函数
5. `*`，数乘，定义为友元函数
6. `*`，矩阵乘法，定义为友元函数

对于 `Matrix<T> a` 和 `Matrix<V> b`，若 `T + V` 的类型为 `U`，则 `a+b` 的返回值类型为 `Matrix<U>`（提示：`decltype`)。回答问题：**为何要定义为友元函数而非成员函数**

3.4 迭代器

要求实现一个 random_access 的迭代器（具体要求见源代码），以及如下函数。

1. `iterator begin()`
2. `iterator end()`
3. `std::pair<iterator, iterator>`

4. `T& operator*() const`，返回当前迭代器指向的数据。

5. `T* operator->() const`，返回迭代器的指针。

6. `subMatrix(std::pair<std::size_t, std::size_t> l, std::pair<std::size_t, std::size_t> r)`. 设 M 是当前矩阵的一个子矩阵，`l` 和 `r` 分别为它的左上角和右下角在原矩阵中的位置，返回 M 的 `begin()` 和 `end()`

3.5 其他

1. `void clear()`，清空当前矩阵，并释放内存空间。

2. `std::size_t rowlength()`，返回行数。

3. `std::size_t columnlength()`，返回列数。

4. `std::pair<std::size_t, std::size_t> size() const`，返回 `size`。

5. （仅 A 班）`void resize(std::size_t n, std::size_t m, T _init = T())`. 保留前 $n*m$ 个元素，若元素不足则拿 `_init` 补充，并重新以行优先方式组成新矩阵。若元素个数相同，则不允许重新开设内存空间。

6. （仅 A 班）`void resize(std::pair<std::size_t, std::size_t> sz, T _init = T())`. 要求同上。

4 说明

1. 整个项目基于 C++14 标准完成，不清楚如何设置的同学可以咨询助教。

2. 编译命令:g++ test.cpp -o test -std=C++14 -O2 -Wall -lopenblas

3. 尽可能项目中少出现 Warning，有助于减少不必要的 bug。

4. A、B 班的评分分别进行，评分标准以对应班级助教组为准。

5 PolicyIterator（选做不做）

5.1 说明

本节内容为 OOP1 作业的选做部分，内容为基于 Policy-Based Class Design 来设计三类迭代器：RowIterator, ColumnIterator , TraceIterator. 分别用于按行遍历，按列遍历以及遍历对角线。请参考后续章节中的简介或者参考书目 *Modern C++ Design*, Andrei Alexandrescu.

此项内容选做不做，**没有** bonus.

5.2 Policy-Based Class Design

此节为 Policy-Based Class Design 的一个简介。

基于 Policy 的设计，是为了解决如下的情况。对于库的设计者，通常会需要在各个方面进行权衡取舍，诸如性能和安全性，是否多线程安全等。为了让库的使用者可以根据需求做出选择，一种解决方法是提供所有的可能用到的类。诸如 `ThreadSafePtr`, `NaivePtr`, `FastPtr` 等等。但是当某一个类的可供选择的特性较多时，不同的选择之间有不同的组合，从而就产生了组合爆炸的问题，如果要提供所有的接口，一方面会产生一定的重复代码，另一方面生产的程序的大小也会相应地膨胀。而基于 Policy 的设计则是为了解决这样情况。

基于 Policy 的设计的宗旨在于，将这些可供选择的特性包装在一些被称为 Policy Class 的类中，而用户在使用的时候，可以通过选择不同的 policy 来组合出所需要的类。在实现上，通常采用 template 来完成。具体可见如下摘录自 *Modern C++ Design* 中的例子。

```
template <class T>
struct OpNewCreator {
    static T* Create() {
        return new T;
    }
};

template <class T>
struct MallocCreator {
    static T* Create() {
        void* buf = std::malloc(sizeof(T));
        if (!buf)
            return nullptr;
        return new(buf) T;
    }
};

template <template<class Created> CreationPolicy>
class WidgetManager : public CreationPolicy<Widget> {
    ...
};
```

```
22
23      // Application Code
24      typedef WidgetManager<OpNewCreator> MyWidgetMgr;
```

5.3 要求

完成迭代器 `PolicyIterator<Policy>` 以及相应的 Policy: `RowIterator`, `ColumnIterator`, `TraceIterator`. 以及相应的 `begin` 和 `end`. 具体内容看一眼测试就好啦 OvO

6 参考资料

大家有什么问题，可以先查一下这些网站：

1. en.cppreference.com/w/
2. www.cplusplus.com
3. stackoverflow.com

另外关于迭代器，可以参考候捷所著的《STL 源码剖析》。
关于 C++11/14 的内容，可以参考 Scott Meyers 所著的《Effective Modern C++》以及之前提到的网站。