高维数组

刷新

题目说明

English Version (/staticdata/problem/2186.96hzmYvZ07xnnK6p.pub/vipMYopKIo2A4xWG.multi-dimensional-array-en.pdf/multi-dimensional-array-en.pd

小红正在写一个高维数组模板类 `template<class T, int N> MultiArray`。其中 T 代表其存储的数据类型，N 代表其维度。现在请你帮助她逐步完善功能，依次完成以下任务：

子任务一：初始化（20 分）

首先，我们需要实现高维数组的如下基本功能：

- `MultiArray()`：生成一个维数为 N 的空数组（每一维大小为 0）
- `MultiArray(const vector<int> &dims, const vector<T> &data = vector<T>())`：给定维数 `dims` 和数据 `data`，生成一个维度为 N 的数组，其中每一维大小分别为 `dims` 的每一位（保证 `dims` 每一位均为正整数且长度不大于 N），其中各元素从地址低到高分别赋值为 `data` 各项（保证 `data` 长度为 `dims` 各位之积）。例如，`MultiArray<int, 2> a({2, 2}, {1, 2, 3, 4})` 即生成一个二维数组 `a`，两个维度大小都是 2，各元素分别赋值为 `a[0][0] = 1, a[0][1] = 2, a[1][0] = 3, a[1][1] = 4`。此外，该方法需要支持如下特殊情况：
  - `data` 为空时（即长度为 0 时，下同），数组各元素初始化赋值为 `T()`；
  - `dims` 为空时，设置第 0 维大小为 `data` 的长度 M，其余维大小均设置为 1；
  - `data` 与 `dims` 不同时为空；
  - `dims` 不为空时但长度小于 N 时，前若干维大小设置为 `dims` 的每一位，缺省维大小均设置为 1
- `void set(const vector<int> &idx, const T &val)`：将数组 `idx` 位置的值设置为 `val`（保证 `idx` 每一位均小于数组对应维大小且长度为 N
- `T& get(const vector<int> &idx)`：返回数组 `idx` 位置的值的引用（对 `idx` 的限定同上）。例如对于上述的二维数组 `a`，若 `idx = {1, 2}` 则返回 `a[1][2]` 的引用
- `vector<int>& get_dims()`：返回一个长为 N 的 `vector<int>&`，代表当前数组各维度的大小

子任务二：基本操作（20 分）

进一步的，我们希望多维数组能够完成如下基本操作：

- 支持 `MultiArray()` 类型的 `==`，`!=`，`=` 运算符，其中：
  - 当且仅当两个 `MultiArray()` 类型数组的各维大小，各元素值均相同时认为两者相等，两者有任一不同时认为两者不相等。将未初始化赋值的数组进行比较是未定义行为，不需要实现
  - 赋值运算符 `=` 仅允许在两个各维度大小相同（或者仅通过 `MultiArray()` 初始化）的高维数组之间使用，等同于对各位置元素同时赋值
- 支持流运算符打印输出整个数组，打印顺序按地址从低到高，示例如下（最后不包含回车，可参考样例）：

```
[0][0][0] = 1
[0][0][1] = 2
[0][0][2] = 3
[0][1][0] = 4
...
[3][4][2] = 114
```

子任务三：迭代器实现（20 分）

实现简易的高维数组迭代器类 `Iterator`，支持如下操作：

- 取值运算符 $*$
- 前后缀 `++` 运算符，`==`，`=`，`!=` 运算符
- `begin()`：返回指向对应高维数组的第0个元素的迭代器
- `end()`：返回指向对应高维数组的最后一个元素之后的迭代器

子任务四：下标访问（20 分）

子任务四、五之间相互独立，你可以跳过子任务四先完成子任务五

不同于 `T& get(const vector<int> &idx)` 方法依照 `vector<int> idx` 访问数据元素，在这一阶段，我们希望直接使用下标访问数据元素。例如，对于 `MultiArray<int, 2> A({2,2},{1,2,3,4})` ，我们需要支持使用 `A[0][2]` 返回数值为 2 的位置的引用，即支持以下代码：

```
MultiArray<int, 2> A({2,2},{1,2,3,4});
A[1][1] = 10;
cout << A[1][1] << endl;
```

子任务五：维度变换（20 分）

最后，我们需要支持高维数组的两种常见维度变换方法：

- `reshape(const vector<int> &dims)`：将高维数组的维度变换为 `dims` （保证 `dims` 的长度为 N）。其中各元素从地址低到高保持不变。如果 `dims` 各位之积和当前高维数组总容量不相同，则忽略此次操作。
- `resize(const vector<int> &dims)`： 将高维数组的维度变换为 `dims` （保证 `dims` 的长度为 N）。对于变化后高维数组的各个位置，如果其对应下标在原高维数组中存在，则其值和原高维数组对应下标处相同；否则，其值初始化为 `T()` 。例如， 对于 `MultiArray<int, 2> a({1, 3, 4}, {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12})` 进行 `a.resize({1, 2, 3})` 操作后，其变化为： `MultiArray<int, 2> a({1, 2, 3}, {1, 2, 3, 6, 7})`
- 同时，在该子任务中，我们需要对内存使用进行检查，必须没有内存泄露才能获得全部的分数

题目要求

给定 `main.cpp` 、 `node.h` 、 `array.h` ，内容见下载链接 (/staticdata/problem/2186.96hzmYvZ07xnnK6p.pub/QXd9JuymTgkrqcQb.download.zip/download.zip)。完成如下要求：

- 不修改 `main.cpp` 、 `node.h` ，其中 `node.h` 为自定义分数类，仅在**子任务五**中作为高维数组的数据类型使用，你不需要关注其具体实现
- 在 `array.h` 中实现高维数组的各项功能，你可以直接在该头文件中实现，亦可实现在其他 `*.cpp` 文件中；我们在下发文件中预定义了 `MultiArray` 类的部分方法，你可以根据需要进一步添加

评分规则

本题的子任务**不使用**捆绑测试。具体地，前 4 个子任务均包含 1 个公开测试点和 1 个隐藏测试点，各占子任务 50% 的分数；第 5 个子任务包含 2 个公开测试点和 2 个隐藏测试点，各占子任务 25% 的分数，其中有 50% 的测试点对内存泄露进行了检查。**所有公开测试点和下发文件完全一致。** 所有测试点中涉及的高维数组规模都足够小，因此你无需刻意关注程序的运行速度。此外，保证所有操作均合法，不需要处理异常情况。

输入输出样例

本题没有输入文件，所有公开测试点对应的 `main.cpp` 文件分别位于下发文件的 `main_1.cpp ~ main_6.cpp` 中，其对应标准输出对应 `1.out ~ 6.out` 。其中公开测试点1 ～ 4 对应子任务一 ～ 四，5 ～ 6 对应子任务五。 在测试时，你可以将对应下发文件重命名为 `main.cpp` 以方便使用统一的 `Makefile` 生成可执行文件。

提示

- **对于子任务四**，赋值操作和流运算需要值具有不同的类型，因此你可能需要实现两个版本的 `[]` 运算符重载
- 部分公开测试点的标准输出可能较长。对于Windows环境，可以使用 `cmd` 窗口的 `fc file1 file2` 命令自动实现两个输出文件的比较；对于Mac和Linux，则可以使用 `diff file1 file2` 命令完成该功能

提交格式

请将 `array.h` 、 `Makefile` 、以及其他你实现的 `*.cpp` 打包成一个 zip 格式的压缩包并上传，提交的其他文件将被忽略。使用的Makefile必须要能生成可执行文件main（不带扩展名）。**注意：你的文件应该在压缩包的根目录下，而不是压缩包的一个子文件夹下，换而言之，解压你提交的压缩包后，应该直接得到一系列 cpp 文件、h 文件等代码文件，而不是一个包含它们的文件夹。评测时，OJ会将提供的文件贴入你的目录下。**

语言和编译选项

#	名称	编译器	额外参数	代码长度限制
0	oop_custom	make		65536 B

递交历史

#	状态	时间
表中没有数据		

当前没有提交权限!