小明的MagicArray

题目描述

刷新

小明最近学习了C++的多态，多态的一个重要意义是为不同类型的对象复用相同的代码。小明尝试设计了一个 `MagicArray` ，这个 `MagicArray` 不保存具体数值，但可以对它及它的元素做一些操作，这些操作之后能够复用到一个同样长度的 `std::vector` 容器中去。

首先为了简单，小明只允许 `arr[0] += i` 这样的操作，其中数组长度固定为1， `i` 为int类型。具体来说：

```
//subtask1
MagicArray arr(1);  // create a MagicArray with length=1
for(int i = 0; i < 10;  i++){
    arr[0] += 1;
}

cout << "#### Test Case 1-1 ####" << endl;
std::vector<int> vec1{0};
arr.apply(vec1); //apply "vec1[0] += 1" for 10 times
cout << vec1[0] << endl; // 10

cout << "#### Test Case 1-2 ####" << endl;
std::vector<int> vec2{5};
arr.apply(vec2); //apply "vec2[0] += 1" for 10 times
cout << vec2[0] << endl; // 15

cout << "#### Test Case 1-3 ####" << endl;
std::vector<double> vec3{1.5}; // You can even use vector<double>
arr.apply(vec3); //apply "vec3[0] += 1" for 10 times
cout << vec3[0] << endl; // 11.5
```

接下来，小明希望 `MagicArray` 能够支持更长的数组。同时为了方便理解， `MagicArray` 能够输出自己记录的操作：

```
//subtask2
MagicArray arr(5);  // create a MagicArray with length=5
for(int i = 0; i < 5;  i++){
    arr[i] += i;
}

cout << "#### TestCase 2: Output Instructions ####" << endl;
static_assert(std::is_abstract<Instruction>(), "Instruction should be an abstract class");
std::vector<Instruction*> &instructions = arr.getInstructions();
for(auto ins : instructions) {
    ins -> output();
}
// arr[0] += 0
// arr[1] += 1
// arr[2] += 2
// arr[3] += 3
// arr[4] += 4

cout << "#### TestCase 2-1 ####" << endl;
std::vector<int> vec1{0, 0, 0, 0, 0};
arr.apply(vec1);
for(auto x : vec1) cout << x << " "; // 0 1 2 3 4
cout << endl;

cout << "#### TestCase 2-2 ####" << endl;
std::vector<double> vec2{5.5, 4.4, 3.3, 2.2, 1.1};
arr.apply(vec2);
for(auto x : vec2) cout << x << " "; // 5.5 5.4 5.3 5.2 5.1
cout << endl;
```

其中 `Instruction` 为抽象类，小明打算为不同的的操作设计不同的 `Instruction` 子类。小明已经写好了该基类的接口。如果有必要，你也可以对该接口做一些修改。

```
class Instruction
{
public:
    virtual void apply(std::vector<int> &vec) = 0;
    virtual void apply(std::vector<double> &vec) = 0;
    virtual void output() = 0;
};
```

接下来，小明希望 `MagicArray` 可以支持更多操作，例如 `a[i] = a[j]` 和 `a[i] += a[j]` 。

```
//subtask3
MagicArray arr(3);  // create a MagicArray with length=3
for(int i = 0; i < 2;  i++){
    arr[i+1] += arr[i];
    arr[i] = arr[i+1];
    arr[i] += i+1;
}

cout << "#### TestCase 3: Output Instructions ####" << endl;
static_assert(std::is_abstract<Instruction>(), "Instruction should be an abstract class");
std::vector<Instruction*> instructions = arr.getInstructions();
for(auto ins : instructions) {
    ins -> output();
}

// arr[1] += arr[0]
// arr[0] = arr[1]
// arr[0] += 1
// arr[2] += arr[1]
// arr[1] = arr[2]
// arr[1] += 2

cout << "#### TestCase 3-1 ####" << endl;
std::vector<int> vec1{0, 0, 0};
arr.apply(vec1);
for(auto x : vec1) cout << x << " "; //1 2 0
cout << endl;

cout << "#### TestCase 3-2 ####" << endl;
std::vector<double> vec2{0.3, 0.2, 0.1};
arr.apply(vec2);
for(auto x : vec2) cout << x << " "; //1.5 2.6 0.6
cout << endl;
```

最后，小明为 `MagicArray` 添加了一个输出操作，例如 `cout << arr[i]`。该输出操作不会立即执行，而是会等到调用 `apply` 时才会进行输出。为了在apply时换行，小明使用了 `arr.endl()` 代替了原有的 `endl`。

```
//subtask4
MagicArray arr(3);  // create a MagicArray with length=3
for(int i = 0; i < 2;  i++){
    arr[i] += arr[i+1];
    arr[i] += i + 1;
    arr[i+1] = arr[i];
    cout << arr[i] << arr.endl(); // arr.endl() is a special object for linebreak
}

cout << "#### TestCase 4: Output Instructions ####" << endl;
static_assert(std::is_abstract<Instruction>(), "Instruction should be an abstract class");
std::vector<Instruction*> instructions = arr.getInstructions();
for(auto ins : instructions) {
    ins -> output();
}
// arr[0] += arr[1]
// arr[0] += 1
// arr[1] = arr[0]
// cout << arr[0]
// cout << endl
// arr[1] += arr[2]
// arr[1] += 2
// arr[2] = arr[1]
// cout << arr[1]
// cout << endl

cout << "#### TestCase 4-1 ####" << endl;
std::vector<int> vec1{0, 0, 0};
arr.apply(vec1);
//6
//9
for(auto x : vec1) cout << x << " "; //6 9 9
cout << endl;

cout << "#### TestCase 4-2 ####" << endl;
std::vector<double> vec2{0.3, 0.2, 0.1};
arr.apply(vec2);
//1.5
//3.6
for(auto x : vec2) cout << x << " "; //1.5 3.6 3.6
cout << endl;
```

- 提示：
  - 你应该自定义一个类型作为 `MagicArray` 中 `operate[]` 返回的元素
  - 对于 `MagicArray` 及其元素的任意操作都应转换为相应的 `Instruction` 派生类，储存在 `MagicArray` 中
- 为了简单考虑，我们对题目做出以下限制：
  - 你只用考虑 `vector<int>` 和 `vector<double>`
  - 元素下标访问保证不会越界

    - 保证 apply 中传入的 vector 长度与 MagicArray 一致
- 我们的 main.cpp 和 Makefile 已经给定，同时我们也给出了 Instruction.h 的基类接口。如果需要，你可以适当修改 Instruction.h 。该 Makefile 将自动的将该目录下的所有的后缀名为 cpp 的程序编译并链接。
    - 特殊地， Makefile 支持测试部分subtask。如果你只想测试 subtask1 ，可以使用 make subtask1 命令。
    - **你可以修改 Instruction.h ，但不能修改 main.cpp 和 Makefile **。无论你是否修改 Instruction.h ，你都应该上传该文件。
- 文件下载地址：下载链接 (/staticdata/1811.j4VXV3kbgUyuChmk.pub/SQhHx4zsldviA38h.download.zip/download.zip)

样例输出

针对我们给出的 main.cpp ，输出如下。

```
#### Test Case 1-1 ####
10
#### Test Case 1-2 ####
15
#### Test Case 1-3 ####
11.5
#### TestCase 2: Output Instructions ####
arr[0] += 0
arr[1] += 1
arr[2] += 2
arr[3] += 3
arr[4] += 4
#### TestCase 2-1 ####
0 1 2 3 4
#### TestCase 2-2 ####
5.5 5.4 5.3 5.2 5.1
#### TestCase 3: Output Instructions ####
arr[1] += arr[0]
arr[0] = arr[1]
arr[0] += 1
arr[2] += arr[1]
arr[1] = arr[2]
arr[1] += 2
#### TestCase 3-1 ####
1 2 0
#### TestCase 3-2 ####
1.5 2.6 0.6
#### TestCase 4: Output Instructions ####
arr[0] += arr[1]
arr[0] += 1
arr[1] = arr[0]
cout << arr[0]
cout << endl
arr[1] += arr[2]
arr[1] += 2
arr[2] = arr[1]
cout << arr[1]
cout << endl
#### TestCase 4-1 ####
6
9
6 9 9
#### TestCase 4-2 ####
1.5
3.6
1.5 3.6 3.6
```

提交格式

你可以提交一个压缩包，其中必须包含 MagicArray.h 和 Instruction.h 。我们会将你提交的文件和我们预先设置好的 main.cpp 、 Makefile 一起编译运行。

注意：你的所有文件应该放在压缩包的根目录下，且文件名中不能包含中文，若提交后发现OJ反馈不正常，请尽快与监考老师联系。

评分标准

我们会有1个样例测试点，即下发的 main.cpp 。另外有1个隐藏测试点，会相应的更改 main.cpp 中内容进行测试。一般来说，如果正确实现了题目要求，并能通过样例测试点（而不是通过某种方法直接输出标准答案），也应该能够通过隐藏测试点。（**若提交后存在问题，请及时联系监考老师。**）

本题按照以上描述分为了4个子任务，每个子任务各占25分。只有你通过了同一个子任务的样例测试点和隐藏测试点，你才能获得该子任务的分数。

- SUBTASK1：保证 MagicArray 长度为1，只有 +=(int) 操作
- SUBTASK2：保证 MagicArray 长度小于100，在SUBTASK1基础上，有输出 Instruction 操作
- SUBTASK3：保证 MagicArray 长度小于100，在SUBTASK2基础上，有 +=(元素) 、 =(元素) 操作
- SUBTASK4：保证 MagicArray 长度小于100，在SUBTASK3基础上，有 cout << 元素 、 cout << MagicArray::string("xx") 操作。

注意你不用同时通过4个子任务再提交，我们会将每一个子任务的代码拆开，分别编译。（但是后一个子任务会依赖于前一个子任务，例如：如果想要第三个子任务正确，必须先保证第一、二个子任务正确。）

考试100%为OJ评分。

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

递交答案

语言和编译选项 oop_custom

```
1
```

提交

文件请拖入编辑器中，或 上传文件