# 小明的list

## 题目描述

刷新

小明最近学了python的list（也就是python中的数组），感觉非常好用，于是也想在C++里实现一个类似功能的类。

小明将这个类命名为MyList，具体来说，需要满足以下的功能：

- 通过append可以向MyList的最后添加元素。
- x[i]可以访问MyList的第i个元素。
- 可以用cout输出MyList中的所有元素。

```
//subtask 1
MyList x;
std::cout << x << std::endl; //[]
for(int i = 0; i < 5; i++)
    x.append(i);
std::cout << x[0] << std::endl; // 0
std::cout << x << std::endl; //[0,1,2,3,4]
```

Python中的对象有一个特性，即所有的对象均为引用。下列代码中，对于y的修改，也会导致x的修改。

```
//subtask 2
MyList y = x;
y[0] += 5;
std::cout << x << std::endl; //[5,1,2,3,4]
```

接下来，小明还想实现slice功能。拿出MyList的任意一段，也可以当做一个新的MyList，较小的那一段被称为原MyList的一个slice。并且对slice的修改也会影响原始的MyList。

```
//subtask 3
MyList z = x(1, 3); //左闭右开选出位置在[1,3)的元素。保证slice左端点不大于右端点。
std::cout << z << std::endl; //[1,2]
z[1] += 5;
std::cout << z << std::endl; //[1,7]
std::cout << x << std::endl; //[5,1,7,3,4]
```

Slice还有一个神奇的特点。如果向slice中append，不仅slice的最后会多出一个新增的元素，原始的List也会在相应位置加入该元素。

```
//subtask 4
MyList r = x(2, 4);
std::cout << r << std::endl; // [7,3]
r.append(-1);
std::cout << r << std::endl; // [7,3,-1]
std::cout << x << std::endl; // [5,1,7,3,-1,4]
```

为了简便，该题的list只存储int，并且可以默认在使用中不出现非法的访问操作。

小明已经完成了部分代码，但是调试一直没法通过。你需要帮助他改好相应的功能，并通过测试程序。 `main.cpp` 为测试程序， `MyList.h` 是小明实现的部分代码。 文件下载地址：下载链接 (/staticdata/2011.VsSeG2zVqbPnpZyy.pub/6zNIuol5JBEuTv0K.download.rar/download.rar)

## 对小明代码的解释

你可以采用小明的代码，**也可以完全不使用小明的代码**，他的代码只作为本题的提示。你只用提交 `MyList.h` ，通过测试即可。

需要注意的是，**小明的已完成的代码里包含一些错误**，你需要进行修改。

这里对小明的设计做一些解释。

观察四个subtask可以发现，所有的操作实际上都是对同一个数组进行了操作。因此小明决定将MyList封装成为一个类似迭代器的类。迭代器拥有储存类的指针和当前的位置。而MyList类同样也有储存类的指针，但记录了起始位置和结束位置。小明将真正的内容储存在std::list中。每个MyList包含一个指针pt指向真正的std::list，left和right代表MyList在std::list的起始终止位置。它们的关系可以参考下图。

其中x, y, z的pt均指向同一std::list。x, y的left指向开头，right指向结尾，所以他们都代表整个序列。z的left指向1，right指向3，所以其内容应该是[1,2] （不包含3，左闭右开区间）。

为了保证内存不泄露，pt使用了stl的动态指针std::shared_ptr。left，right记录std::list的位置信息，使用的是std::list::iterator。（使用std::list的原因是避免迭代器的失效问题。）

提示：涉及stl的问题往往会带来很多编译错误，从第一条error看起会比较容易。

# 提交格式

你只能提交头文件 `MyList.h`，我们会将你提交的文件和我们预先设置好的 `main.cpp` 一起编译运行。

你应该将你的文件打包成一个zip压缩包并上传。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。

# 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，如果你能正确实现通过样例测试点，也应该能够通过隐藏测试点。

本题按照以上描述分为了4个subtask，每个subtask各占25分。只有你通过了样例测试点和隐藏测试点的同一个subtask，你才能获得该subtask的分数。

**注意你不用同时通过4个subtask再提交**，我们会将每一个subtask的代码拆开，分别编译。（但是后一个subtask会依赖于前一个subtask。）

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` | | 1048576 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