题目描述

本次题目要求你构建一个演唱会购票系统，实现购票、退票等功能。为了实现这些功能，你一共需要实现4个类：座位类(Seat)，普通座位类(RegularSeat)，VIP座位类(VIPSeat)，场馆类(Gym)。其中RegularSeat类和VIPSeat类是Seat类的派生类，表示两种类型的座位（两种座位的差别体现在价格的不同）；Gym类表示演唱会场馆。

- Seat类
  - 你需要在自己定义的Seat类中至少实现以下接口

```
// 构造函数，初始化Seat对象
Seat::Seat(int row, int col)
/* 说明
1）row表示座位所在行，col表示座位所在列
*/

// 成员函数，返回座位所在行
int Seat::GetRow()

// 成员函数，返回座位所在列
int Seat::GetCol()

// 成员函数，重载操作符 >
bool Seat::operator > (const Seat& b)
/* 说明
1）比较两个座位所在行
2）对于座位A>座位B，若A在B后排，则返回true，否则返回false
3）不需要考虑座位行列重复的情况
*/

// 成员函数，重载操作符 ==
bool Seat::operator == (const Seat& b)
/* 说明
1）比较两个座位所在行
2）对于座位A==座位B，若A和B在同一行，则返回true，否则返回false
3）不需要考虑座位行列重复的情况
*/

// 成员函数，重载操作符 -
int Seat::operator - (const Seat& b)
/* 说明
1）比较同一行两个座位相差几个座位
2）对于座位A-座位B，若A和B不在同一行，则返回 -1
3）若A和B在同一行，则返回A和B所在列差值的绝对值
```

```
4) 不需要考虑座位行列重复的情况
*/
```

- RegularSeat类
  - RegularSeat类是Seat类的派生类，你需要在自己定义的RegularSeat类中具有以下接口

```
// 构造函数，初始化RegularSeat对象
RegularSeat::RegularSeat(int row, int col)

//  成员函数(继承自基类)，返回该座位的所在行
int RegularSeat::GetRow()

//  成员函数(继承自基类)，返回该座位的所在列
int RegularSeat::GetCol()

// 成员函数，返回该座位的票价
int RegularSeat::Price()
/* 说明
普通座位票价返回 100
*/
```

  - 接口调用示例

```
// 创建对象
RegularSeat seat1(3,4);

// 查询座位所在行
seat1.GetRow();

// 查询座位所在列
seat1.GetCol();

// 返回票价
seat1.Price();
```

- VIPSeat类
  - VIPSeat类是Seat类的派生类，你需要在自己定义的VIPSeat类具有以下接口

```
// 构造函数，初始化VIPSeat对象
VIPSeat::VIPSeat(int row, int col)

// 成员函数(继承自基类)，返回该座位的所在行
int VIPSeat::GetRow()
```

```
// 成员函数(继承自基类)，返回该座位的所在列
int VIPSeat::GetCol()

// 成员函数，返回该座位的票价
int VIPSeat::Price()
/* 说明
VIP座位票价返回 500
*/
```

- 接口调用示例

```
// 创建对象
VIPSeat seat2(1,1);

// 查询座位所在行
seat2.GetRow();

// 查询座位所在列
seat2.GetCol();

// 返回票价
seat2.Price();
```

- Gym类
  - 你需要在自己定义的Gym类中至少实现以下接口

```
// 构造函数，初始化Gym对象
Gym::Gym(int RegularSeatNum, int VIPSeatNum)
/* 说明
1) (RegularSeatNum, VIPSeatNum)为该场馆的普通座位数和VIP座位数容量上限
2) RegularSeatNum和VIPSeatNum均大于等于0
*/

// 成员函数，添加座位
bool Gym::AddSeat(Seat* seat, int type)
/* 说明
1) 根据type判断是普通座位还是VIP座位：0表示普通座位，1表示VIP座位
2) 若添加普通座位，当场馆普通座位数已满时，则添加失败，场馆中已有座位保持不变，返回false
3) 若添加VIP座位，当场馆VIP座位数已满时，则添加失败，场馆中已有座位保持不变，返回false
4) 当添加的座位对应的座位数仍有剩余容量时，则添加成功，返回true
5) type和座位的行列确定一个座位，当添加的座位重复，添加失败，场馆中已有座位保持不变，返回false
*/
```

```
// 成员函数，删除座位
bool Gym::DeleteSeat(Seat* seat, int type)
/* 说明
1）根据type判断是普通座位还是VIP座位：0表示普通座位，1表示VIP座位
2）若删除普通座位，当场馆无该普通座位，则删除失败，场馆中已有座位保持不变，返回false
3）若删除VIP座位，当场馆无该VIP座位，则删除失败，场馆中已有座位保持不变，返回false
4）当删除的座位存在时，则删除成功，返回true
5）type和座位的行列确定一个座位
*/

// 成员函数，购票
Seat* Gym::Buy(int row, int col)
/* 说明
1）若该位置的座位已经被购买或者场馆中不存在这个座位，则购买失败，返回NULL
2）否则返回该座位的指针
3）座位的行列确定一个座位
*/

// 成员函数，退票
Seat* Gym::Refund(int row, int col)
/* 说明
1）若该位置的座位未被购买或者场馆中不存在这个座位，则退票失败，返回NULL
2）否则返回该座位的指针
3）座位的行列确定一个座位
*/

// 成员函数，计算该场馆的收入
int Gym::Income()
/* 说明
只有卖出去的座位才算收入
*/
```

- 接口调用示例

```
// 创建对象
Gym gym1(3,2);
RegularSeat seat3(3,5);
RegularSeat seat4(3,6);
RegularSeat seat5(3,7);
VIPSeat seat6(1,2);
VIPSeat seat7(1,3);

int ans1 = seat3 - seat5; // ans1 == 2
int ans2 = seat6 - seat4; // ans2 == -1
bool ans3 = (seat4 == seat5); // ans3 == true
bool ans4 = (seat5 > seat7);  // ans4 == false

// 添加座位
```

```
gym1.AddSeat(&seat3,0);
gym1.AddSeat(&seat4,0);
gym1.AddSeat(&seat5,0);
gym1.AddSeat(&seat6,1);
gym1.AddSeat(&seat7,1);

gym1.AddSeat(&seat7,1); // false

// 删除座位
gym1.DeleteSeat(&seat5,0);
gym1.DeleteSeat(&seat6,1);

gym1.DeleteSeat(&seat6,1);  // false

// 购票
gym1.Buy(3,6);
gym1.Buy(1,3);

gym1.Buy(1,3);  // NULL

// 退票
gym1.Refund(1,3);

gym1.Refund(1,3); // NULL

// 计算该场馆的收入
gym1.Income();
```

注意⚠️

- 请正确处理头文件和实现文件之间的关系，文件、函数的命名严格按照给定要求，注意大小写。
- 将4个文件(Seat.cpp, Seat.h, Gym.cpp, Gym.h)打包成ZIP压缩包上传(ZIP包中不要包含文件夹或者其他文件)。
- 请不要在你提交的代码中包含main函数。