题目描述

在数学中，由若干个单项式相加组成的代数式叫做多项式。多项式可采用多种数据结构进行实现。 现有一个基类 Polynomial， 如下所示：

```
## “Polynomial.h”
class Polynomial {
public:
    // 增加一个系数为coef，指数为power的项
    // 若原有的多项式已有指数为power的项，则修改其系数为 new_coef = old_coef + coef，否则新增一个系数为coef，指数为power的项
    // 注意：若修改后的coef为0，则在多项式中应该删除这个项
  virtual void addTerm(int coef, int power) = 0;
    // 第i项表示实际存储的第i个单项式，i从0开始，测试样例给出的i不会越界
    // 设置第i项的系数
  virtual void setIthCoe(int i, int coef)= 0;
    // 设置第i项的指数
  virtual void setIthPow(int i,int power) = 0;
    // 获取第i项的系数
  virtual int getIthCoe(int i) = 0;
    // 获取第i项的指数
  virtual int getIthPow(int i) = 0;
    // 获取多项式长度
  virtual int getSize() = 0;
};
```

现有以下两个函数模板，需要完成上述代码的实现，使得其能够正确运行以下两个函数

```
template<class T> void sortPoly(T *polys) {// 实现多项式按照指数降序排序
    int n = polys->getSize();
    while (n>1) {
        int i_min = 0;
        for (int i = 1; i<n; i++)
            if (polys->getIthPow(i) < polys->getIthPow(i_min))
                i_min = i;
        if (i_min != n-1) {
            int p = polys->getIthCoe(i_min),
            q = polys->getIthPow(i_min);
            polys->setIthCoe(i_min, polys->getIthCoe(n-1));
            polys->setIthPow(i_min, polys->getIthPow(n-1));
            polys->setIthCoe(n-1, p);
            polys->setIthPow(n-1,q);
        };
        n--;
    }
}
```

```
template<class T> void print(T *polys) {// 实现多项式的打印功能
    unsigned int n = polys->getSize();
    for (int i = 0; i < n; i++)
        cout << polys->getIthCoe(i) << " " << polys->getIthPow(i) << endl;
}
```

注意：这两个模板函数在提交时由我们给出，不需要包含在提交文件中（**本地测试可以实现调用，提交的代码中不需要包含**）。

实验中需要实现基于链表与数组实现的多项式，为了简化实现，多项式中**没有**系数为 0 的项。

1. 多项式的数组实现

   使用如下的数据结构实现一个基于数组的多项式类

```
struct Item {// 多项式的单个项数据结构
    int coef;
    int power;
    Item() {}
    Item(int a, int b) { coef = a; power = b; }
};
```

基于`Item`，你需要实现`ArrayPoly`类，该类继承`Polynomial`类，类定义如下：

```
class ArrayPoly: public Polynomial {
 private:
    // 使用Item的数组实现，不允许使用stl中的模版实现
    Item* item_array;
 public:
    // 默认构造函数
    ArrayPoly();

    // 拷贝构造函数
    ArrayPoly(ArrayPoly const& a);

    // 析构函数
    ~ArrayPoly();

    // 赋值操作符重载
    ArrayPoly& operator = (const ArrayPoly& p);

    // TODO: 实现其他你需要实现的代码
};
```

2. 多项式的链表实现

使用如下的结构作为链表节点，实现一个基于链表的多项式类

```
struct Node {
    int coef;
    int power;
    Node *m_pre, *m_next;

    Node(int coef, int power) {
        m_pre = nullptr;
        m_next = nullptr;
        coef = coef;
        power = power;
    }
};
```

基于`Node`，你需要实现`ListPoly`类，该类继承`Polynomial`类，类定义如下：

```
class ListPoly: public Polynomial {
  private:
    // 指向链表头的指针
    Node* head;
  public:
    // 默认构造函数
    ListPoly();

    // 析构函数，释放申请空间
    ~ListPoly();

    // 拷贝构造函数
    ListPoly(ListPoly const &polynomial);

    // 赋值操作符重载
    ListPoly& operator = (const ListPoly& p);

    // TODO：实现其他你需要实现的代码
};
```

注意：请严格按照上述数据结构实现，我们会检查你们的代码，不符合要求的得不到相应的分数。

调用示例

```
Polynomial* poly1 = new ArrayPoly();
poly1->addTerm(3, 4);
poly1->addTerm(7, 8);
poly1->addTerm(8, 3);
unsigned n = ap1->getSize();
sortPoly(poly1);
print(poly1);
```

```
delete poly1;
// 输出:
//        7 8
//        3 4
//        8 3

ListPoly lp1;
lp1.addTerm(5, 6);
lp1.addTerm(7, 8);
lp1.addTerm(3, 3);
ListPoly lp2 = lp1;
sortPoly(&lp1);
print(&lp2);
// 输出:
//        5 6
//        7 8
//        3 3

ArrayPoly* ap1 = new ArrayPoly();
ap1->addTerm(2, 4);
ap1->addTerm(5, 4);
ap1->addTerm(8, 3);
ap1->setIthCoe(1, 1);
ArrayPoly ap2(*ap1);
delete ap1;
print(&ap2);
// 输出:
//        7 4
//        1 3

ListPoly lp3;
lp3 = lp2;
lp2.setIthCoe(2, 2);
print(&lp3);
// 输出:
//        5 6
//        7 8
//        3 3
```

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5个文件(`Polynomial.h`, `ArrayPoly.h`, `ArrayPoly.cpp`, `ListPoly.h`, `ListPoly.cpp`)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。