1 链表的实现

1.1 题目描述

链表和数组都是可以存储序列的数据结构，然而，链表相较于数组而言，其增加元素、删除元素操作却只有 O (1) 的复杂度，也因此往往成为存储那些需要经常增删改查数据的理想的选择。在本次实验中，你需要实现一个简单的货物管理系统，其接口定义如下：

```
  // this is GoodsList.h
#include <string>
using namespace std;
class Goods{
private:
  string name;
  int nums;
public:
  Goods(string name,int nums);
 // 打印货物名称及其数量，格式为 name,nums
  void show();
  ~Goods();
};


class GoodsList{
  private:
     struct Node{
       Goods* goods;
       Node* next;
     } *list;

  public:
```

```
// 构造函数，初始化一个带头结点的链表，返回其头结点，
    GoodsList();
// 返回链表中当前的货物种类
    int count();
// 向当前货物链表中添加一个新的货物
// 如果链表中已有该种类的货物，则在原始货物数量的基础上增加新的数量，否则新建一个结点
    void insert(Goods* goods);
// 从当前货物链表中移除 nums 件名为 name 的货物
// 若移除后该货物数量为 0，则删除该货物结点；
// 若链表中该货物的数量小于要移除的数量，则打印"Not enough goods!\n"，并保持链表不变
    void remove(string name,int nums);
// 打印当前链表中的货物及其数量，格式为 name,nums，每条记录之间以空格分隔，以换行符结尾
    void show();
// 对链表中的货物按数量多少进行排序
    void sort();
//删除链表中所有货物，并释放内存空间，保留头结点，并将头结点指向 nullptr
    void clear();
// 析构函数
    ~GoodsList();
}
```

1.2 示例调用

```
GoodsList list;
Goods* good1 = new Goods("apple",500);
Goods* good2 = new Goods("pear",100);
Goods* good3 = new Goods("peach",150);
list.insert(good1);
list.insert(good2);
list.insert(good3);
list.show();
```

```
list.remove("apple",100);
list.show();
list.remove("apple",1000);
list.show();
cout << list.count() << endl;
list.clear();
cout << list.count() << endl;
```

输出：

```
apple,500 pear,100 peach,150
apple,400 pear,100 peach,150
Not enough goods!
apple,400 pear,100 peach,150
3
0
```

1.3 注意事项

- 下载的压缩包中已经包含了 *GoodsList.h* 文件，你可以在这份文件基础上增加新的函数，但不要更改原始的函数调用接口，否则将无法通过测试。
- 注意内存安全, 避免内存泄漏
- 提交时请将 *GoodsList.h*, *GoodsList.cpp* 打包成为 zip 包，注意文件编码格式为 $utf-8$
- 注意不要在提交的文件中包含 *main* 函数