题目 A（操作符重载）（20 分）

一. 题目描述

实现一个元素为 int 的集合类 Set，要求包含以下接口

接口	功能	说明
Set()	构造函数	
Set(const Set &set)	构造函数	
Set(int elements[], int length)	构造函数	数组中元素未经排序且可能包含重复元素。
~Set()	析构函数	
void add(int element)	添加元素	
bool erase(int element)	删除元素	删除成功返回 true； 不包含该元素返回 false。
bool contains(int element)	包含元素	判断是否包含某个元素
int size()	集合大小	
ostream &operator<<(ostream &os, const Set &rset)	输出元素	升序输出集合中所有元素。若集合只包含-2、-1、0、1、2 五个元素，则输出如下所示，元素之间用**空格**分隔。 **-2 -1 0 1 2** 若集合为空集，则输出为 **Empty**
Set operator+(const Set &lset, const Set &rset)	重载+	并集运算
Set &operator+=(const Set &rset)	重载+=	在本集合中加入 rset 包含的元素
Set operator-(const Set &lset, const Set &rset)	重载-	差集运算
Set &operator-=(const Set &rset)	重载-=	在本集合中删除 rset 包含的元素
Set operator&(const Set &lset, const Set &rset)	重载&	交集运算
Set operator|(const Set &lset, const Set &rset)	重载|	并集运算
bool operator==(const Set &lset, const Set &rset)	重载==	判断两个集合中元素是否完全相同
bool operator!=(const Set &lset, const Set &rset)	重载!=	
bool operator>(const Set &lset, const Set &rset)	重载>	判断 lset 是否真包含 rset
bool operator>=(const Set &lset, const Set &rset)	重载>=	判断 lset 是否包含 rset
bool operator<(const Set &lset, const Set &rset)	重载<	判断 lset 是否真包含于 rset
bool operator<=(const Set &lset, const Set &rset)	重载<=	判断 lset 是否包含于 rset
void *operator new(size_t size)	重载 new	
void operator delete(void *p, size_t size)	重载 delete	

二．调用示例

示例一	

调用代码	int elements[] = {-1, 2, 0, 3, -5, 3};
	int length = sizeof(elements)/sizeof(int);
	Set set(elements, length);
	std::cout << set.size() << endl;
	std::cout << set.contains(5) << endl;
	std::cout << set << endl;
	set.add(6);
	std::cout << set << endl;
	std::cout << set.erase(0) << endl;
	std::cout << set << endl;
	std::cout << set.erase(1) << endl;
	std::cout << set << endl;

期望输出	5 0 -5 -1 0 2 3 -5 -1 0 2 3 6 1 -5 -1 2 3 6 0 -5 -1 2 3 6
示例二	

调用代码	//set1 包含-2,-1,0,1,2
	//set2 包含 1,2,3,4,5
	std::cout << set1+set2 << endl;
	std::cout << set1-set2 << endl;
	std::cout << (set1|set2) << endl;
	std::cout << (set1&set2) << endl;
	set1 += set2;
	std::cout << set1 << endl;
	set1 -= set2;
	std::cout << set1 << endl;

期望输出	-2 -1 0 1 2 3 4 5 -2 -1 0 -2 -1 0 1 2 3 4 5 1 2 -2 -1 0 1 2 3 4 5 -2 -1 0
示例三	

调用代码	//set1 包含-2,-1,0,1,2
	//set2 包含 0,1,2
	//set3 为空集

	std::cout << (set1==set2) << endl;
	std::cout << (set1!=set2) << endl;
	std::cout << (set1>set2) << endl;
	std::cout << (set1>=set2) << endl;
	std::cout << (set1<set2) << endl;
	std::cout << (set1<=set2) << endl;
	std::cout << (set1>set3) << endl;

期望输出	0 1 1 1 0 0 1

三．注意事项

1. 除重载的<<操作符外，其他所有接口**不需要**进行任何输出操作。
2. **不允许使用 STL**，OJ 提交结束后会人工抽查代码，被发现使用会酌情扣分。
3. **new 和 delete 的重载会采用人工检查代码的形式给分**，作为本次作业的简答题，需要在程序里用**注释**对自己的实现进行辅助说明。

四．提交要求

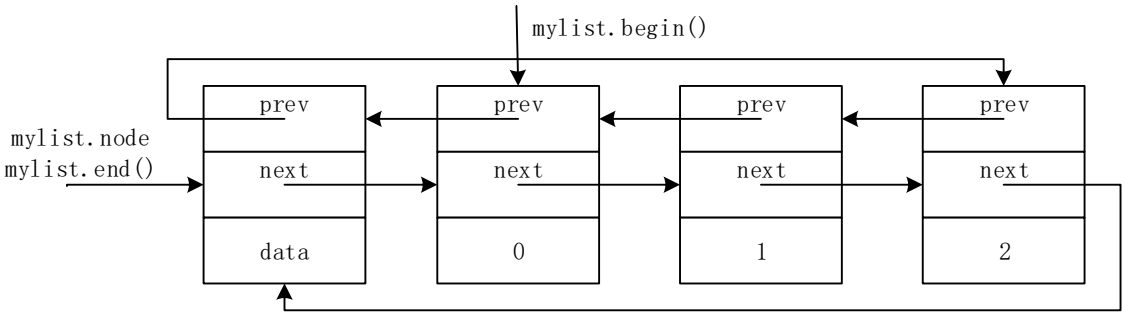
1. 提交 2 个文件：Set.h 和 Set.cpp，直接打包为 zip 格式压缩包，不要存在多一层的目录。
2. 请将方法声明放在.h 文件中，方法实现放在.cpp 文件中。
3. 实现代码请严格按照给定的接口名字，否则不能通过编译。
4. 提交代码中不要包含 main 函数，否则不能通过编译。
5. 严格按照要求的功能实现输出，不要尝试进行其他输入输出活动，否则不能通过测试。

(出题人：殷迪)

题目 B（STL）（20 分）

一. 题目描述

STL 中的 list 是一个双向循环链表容器，它能够存放各种类型的对象。要求实现一个简易 List 类。List 的一个实例如下所示



List 的接口定义如下：

```
template    <class T>
struct _List_node {
    _List_node<T>* prev; //前驱节点
    _List_node<T>* next; //后继节点
    T data; //数据
};


template <class T>
struct _List_iterator {
    _List_node<T>* _p_node;
    _List_iterator(_List_node<T>* x): _ p_node(x) {};
    ~_List_iterator() {};
    //TODO  重载  ==
    //TODO  重载  !=
    //TODO  重载  前置++
    //TODO  重载  后置++
    //TODO  重载  前置--
    //TODO  重载  后置--
    //TODO  重载  *  取元素，返回 data
};


Template <class T>
class List {
public:
    typedef _List_node<T> link_type;
    typedef _List_iterator<T> iterator;
```

```
    List(); //初始化 List，申请入口节点空间
    ~List(); //回收所有节点空间，包括入口节点
    bool empty(); //是否为空，为空返回 true，否则返回 false
    iterator begin(); //返回第一个元素的迭代器
    iterator end(); //返回最后一个元素后面一个位置的迭代器，即[begin, end)

    //在 it 前插入一个元素，返回的迭代器指向插入元素
    iterator insert(iterator it, T e);

    //删除 it 指向的元素，返回的迭代器指向删除元素的下一个元素
    iterator erase(iterator it);
private:
    link_type* node; //list 入口
}
```

二．调用示例

示例一：

```
List<int> l;
List<int>::iterator it = l.begin();
it = l.insert(it, 0);
it = l.insert(it, 1);
it = l.insert(it, 2);
++it;
it = l.erase(it);
for(it = l.begin(); it != l.end(); it++){
    cout << *it << endl;
}
```

结果：

```
2
0
```

示例二：

```
List<char> l;
l.insert(l.end(), ‘a’);
l.insert(l.end(), ‘b’);
l.insert(l.end(), ‘c’);
l.insert(l.end(), ‘d’);
List<char>::iterator it = l.erase(l.begin());
it = l.erase(it);
for(; it != l.end(); ++it){
    cout << *it << endl;
}
```

结果：

c
d

三．注意事项

要求使用类模板，**支持所有的基本的数据类型，不要使用 STL**。

四．提交要求

1. 提交一个源码文件，List.h，直接打包成 zip 压缩格式的压缩包。不要添加任何其他目录。
2. 实现和声明都写在 h 头文件中，文件编码格式为 utf-8。
3. 请严格按照给定的接口进行编码，否则无法调用测试用例。
4. 提交的源码文件中，不能包含 main 函数，否则无法编译通过。

(出题人：高少华)

题目 C（继承）（40 分）

一. 题目描述

模拟实现一个对共享单车的管理系统，其间需要设计实现进行系统管理的 BikeSystem 类，以及表示所有类型共享单车的 Bike 类，该 Bike 类包含两个派生类：OfoBike、HelloBike 类。具体要求如下：

1. 表示共享单车的基类 Bike 类，需要包含如下属性以及接口：
   (1) 单车编号：serial ，字符串类型；
   (2) 占用表示：occupied，布尔类型，初始化为 false；
   (3) 单次使用累计骑行距离：distance，整型，描述的是一次解锁到上锁之间的移动距离，初始化为 0；
   (4) 当前位置横坐标：x，整型，范围[0,100]；
   (5) 当前位置纵坐标：y，整型，范围[0,100]；
   (6) 骑行舒适程度：comfort，整型，范围[1,10]；
   (7) 需要设计接口
      (a) bool unlock(); //解锁单车，若单车空闲(occupied=false),则解锁成功，并设置 occupied=true；否则为解锁失败。
      (b) int lock();// 对单车上锁，如单车在使用中，即 occupied=true，则上锁成功并设置 occupied=false，distance=0，返回使用累计费用；否则上锁失败，返回-1。
      (c) int move_to(int destination_x, int destination_y); //单车从当前位置出发到此目标位置，打印“from **x,y** to **destination_x,dextination_y**”，更新单车中的位置，并返回累计骑行距离 distance。
      (d) int compute_cost();//根据累计骑行距离计算费用，返回该费用。
2. 实现 Bike 的派生类 OfoBike，并需要满足如下要求:
   (1) 构造函数：OfoBike::OfoBike(const char* id, int location_x, int location_y, int comfort_rating);//参数依次为：单车编号，当前位置横坐标，当前位置纵坐标，骑行舒适程度。
   (2) bool unlock(); // 如果解锁成功，则打印“unlock 单车编号”，返回 true；如果解锁失败，打印“fail to unlock 单车编号”,返回 false。
   (3) int lock(); //如果上锁成功，则打印“lock 单车编号”,返回费用；如果操作失败，则打印“单车编号 has been locked”，返回-1。
   (4) int compute_cost():// ofo 单车的车速设置为 21 单位长度/小时，骑行时间向上取整，费用为 2 元/小时，同时由于 ofo 骑行优惠，可以减免 7 元。
3. 实现 Bike 的派生类 HelloBike，并需要满足如下要求：
   (1) 增添一个属性：电子锁编码, 字符串类型(如“HelloBike_elock_89757”)
   (2) 构造函数： HelloBike::HelloBike(const char* id, const char* elock_id, int location_x, int location_y, int comfort_rating);//参数依次为：单车编号，电子锁编码，当前位置横坐标，当前位置纵坐标，骑行舒适程度。
   (3) bool unlock(); // 如果解锁成功，则打印“unlock 单车编号 电子锁编码”，返回 true；如果解锁失败，打印“fail to unlock 单车编号 电子锁编码”
   (4) int lock(); //如果上锁成功，则打印“lock 单车编号 电子锁编码”,返回费用；如果操作失败，则打印“单车编号 电子锁编码 has been locked”，返回-1；

(5) int compute_cost():// HelloBike 的车速设置为 25 单位长度/小时，骑行时间向上取整，费用为 2 元/小时，同时由于 HelloBike 骑行优惠，可以减免 5 元。

4. 实现 BikeSystem 类，统一管理以上涉及的共享单车类；本题 BikeSystem.h 头文件中已经定义所需方法，除了问题中的 search 方法，其他都已经在 BikeSystem.cpp 中实现，只需要完成 **BikeSystem.cpp** 中寻找最近可用单车的 **search** 方法实现。

BikeSystem 类说明：

(1) Bike** bikes；//单车列表

(2) int size；//系统中当前的单车数量

(3) int capacity; //系统支持管理的单车数量上限

(4) int BikeSystem::search(const char* user_name, int start_x, int start_y, int acceptable_comfort_rating); // 参数说明：user_name 用户名，start x, start y 为当前位置，acceptable_comfort_rating, 骑行舒适程度; 方法说明：从成员 bikes 数组中找到满足条件的单车，查找成功则返回单车在数组中的下标，**失败则返回-1**。查找条件：单车无人使用，且舒适度大于等于 acceptable_comfort_rating。如果有多个满足条件的单车，返回距离（start_x,start_y）近的；如果有多辆满足，返回舒适度最高的；如果上述条件都相同，返回在 bikes 数组中下标最小的。查找成功，则输出“用户名 selected 单车编号”；失败则输出 “用户名 found no bikes available”。

二. 调用示例

测试代码：

```
Bike* tempBike=new OfoBike("ofo10", 40,15,4);
tempBike->unlock(); //输出"unlock ofo10"
tempBike->unlock(); //输出 "fail to unlock ofo10"
cout<<tempBike->lock()<<endl; //输出 "lock ofo10" "0"
cout<<tempBike->lock()<<endl; //输出 "ofo10 has been locked" "-1"

Bike* tempBike=new OfoBike(“ofobike09",15,55,3); //test ofobike move_to, cost output
tempBike->unlock();
tempBike->move_to(90,85);
cout<<tempBike->lock()<<endl;

BikeSystem bike_system; // test search
bike_system.add(new OfoBike("ofo05", 30, 60, 9));
bike_system.add(new HelloBike("hellobike01", "hellobike_elock15", 0, 0, 4));
bike_system.add(new OfoBike("ofo01", 100, 100, 5));
bike_system.add(new OfoBike("ofo02", 90, 0, 6));
bike_system.add(new OfoBike("ofo03", 30, 70, 7));
bike_system.add(new OfoBike("ofo04", 70, 0, 8));

int idx1 = bike_system.search("xiaoming", 50, 50, 5);
// 查找成功，输出"xiaoming selected ofo05",返回 idx1=0
int idx2 = bike_system.search("xiaohong", 50, 50, 9);
// 查找成功，输出"xiaohong selected ofo05",返回 idx1=0
```

```
    bike_system.unlock_bike(idx1); // 解锁成功，输出"unlock ofo05"
    bike_system.unlock_bike(idx2); // 解锁失败，输出"fail to unlock ofo05"
    bike_system.ride_record(idx1, 100, 100); // 输出"from 30,60 to 100,100"
    bike_system.ride_record(idx1, 100, 80); // 输出"from 100,100 to 100,80"
    int cost = bike_system.lock_bike(idx1); //锁车成功，输出"lock ofo05"，返回 cost=7
    cout<<cost<<endl;
```

测试输出：

```
unlock ofo10
fail to unlock ofo10
lock ofo10
0
ofo10 has been locked
-1
unlock ofobike09
from 15,55 to 90,85
lock ofobike09
3
xiaoming selected ofo05
xiaohong selected ofo05
unlock ofo05
fail to unlock ofo05
from 30,60 to 100,100
from 100,100 to 100,80
lock ofo05
7
```

三. 注意事项

1. 合理设置各属性成员的访问控制。
2. 注意此题中对 C++的继承/动态绑定等机制的考察使用。
3. 注意题目中涉及到的方法（unlock,lock,move_to,search）的输出要求,对涉及到的属性进行正确的替换。
4. 样例中给出主要是对 OfoBike 的测试，请考虑与 HelloBike 的区别并自我安排测试。
5. 距离使用曼哈顿距离 distance[(x1,y1),(x2,y2)]=|x1-x2|+|y1-y2|。
6. 费用价格非负。
7. 不考虑在无车可用时进行 search 的情况。
8. 注意各个类的 include，否则会出现重复定义错误。
9. 请不要改动增删已有的 BikeSystem 类中已经实现的 api，只需要在 BikeSystem.cpp 中实现寻找最近可用单车的 search 方法，如果 search 需要调用其它基本库，请确认是 g++编译器下的基本库。否则会导致测试不通过。

四. 提交要求

1. 提交 8 个文件，Bike.h、Bike.cpp、HelloBike.h、HelloBike.cpp、OfoBike.h、OfoBike.cpp、BikeSystem.h、BikeSystem.cpp，直接打包为 zip 格式压缩包，不要存在多一层的目录。
2. 请将方法声明放在.h 文件中；方法实现在.cpp 文件中。
3. 实现代码请严格按照给定的接口名字，否则不能通过编译。
4. 提交代码中不要包含 main 函数，否则不能通过编译。
5. 严格按照要求的功能实现输出，不要尝试其他输入输出活动，否则不能通过测试。

(出题人：鲍宇)

题目 D（String）（20 分）

一．题目描述

命令行界面下的工作目录的模拟:在 Linux 的环境下的命令行界面中，存在一个当前工作目录，可以通过命令 pwd 来进行查看，若需要转换到其它的工作目录，可以通过 cd 命令进行切换。如下图所示:

```
user@xz:~$ pwd
/home/user
user@xz:~$ cd Documents/
user@xz:~/Documents$ pwd
/home/user/Documents
```

现在，具体要求如下:

实现一个 Shell 类，接口要求如下:
Public:
Shell(string homedir="/");//构造函数，homedir 表示 Shell 的创建时的当前工作目录，也表示 home 目录。若没有参数，则当前工作目录为 root 目录。
~Shell();//析构函数
void cd(string path="");//将当前工作目录切换到 path 指定的目录，参数 path 可以表示相对路径，也可以表示绝对路径。若没有参数，则表示切换至 home 目录
void pwd();//输出 Shell 的当前工作目录
void distance(string path);//计算当前工作目录，到 path 指定的目录的距离并且输出,**距离的计算见注意事项**
Private:
//自行定义需要的变量

二．调用示例(由于编码问题，最好不要直接复制下面的代码)

示例 1

```
int main(){
    Shell *s = new Shell();
    s->pwd();
    delete s;
}
```

结果:
/

示例 2

```
int main(){
    Shell *s = new Shell();
    s->pwd();
```

```
        s->cd(“~/user/name/..”);
        s->pwd();
        s->cd(“/usr/local/bash”);
        s->cd(“./../.../../hello/./”);
        s->pwd();
        s->cd(“hello/./hello”);
        s->pwd();
        return 0;
}
```

结果:

```
/
/user/
/usr/local/hello/
/usr/local/hello/hello/hello
```

示例 3

```
Int main(){
        Shell *s = new Shell();
        s->distance("./Hello/World/Bash");
        s->distance("/local/user/name");
        delete s;
}
```

结果:

```
3
3
```

示例 4

```
int main(){
        string homedir = “/home/user/Documents”;
        Shell *s = new Shell(homedir);
        s->pwd();
        s->cd(“/”);
        s->cd(“~/user/name/..”);
        s->pwd();
        s->cd(“/usr/local/inclue/”);
        s->pwd();
        s->distance(“./usr/local/inclue”);
        return 0;
}
```

结果:

```
/home/user/Documents/
/home/user/Documents/user/
/usr/local/inclue/
3
```

三．注意事项

1. 测试样例保证 homedir 为绝对路径,并且 homedir 和 path 不会出现连续的\
2. 目录名字包含除了空格、引号、/、~以外的所有可能符号
3. 注意 root 目录和 home 目录两者之间的区别
4. .表示当前目录
   ..表示上一级目录
   ~表示 home 目录
5. 每一个输出结果最后面请加上/,详细见实例
6. **距离的计算，假设当前路径为/a/b/c,path 的值为/a/d/c,那么从当前路径到 path 需要执行 cd ../../d/c,最少需要四次切换，故距离为 4.**

四．提交要求

1. 提交一个源码文件 Shell.h，直接打包成 zip 压缩格式的压缩包。不要添加任何其他目录。
2. 实现和声明都写在.h 头文件中，文件编码格式为 utf-8。
3. 请严格按照给定的接口进行编码，否则无法调用测试用例。
4. 提交的源码文件中，不能包含 main 函数，否则无法编译通过。

(出题人：何泽昊)