定义Set类，继承自List类。Set类将使用List的基础结构来存储元素，但添加额外的方法来执行集合操作。

template <typename T>

class Set : public List<T> {

public:

// 构造函数和析构函数

Set() : List<T>() {}

~Set() {}

// 集合的差（差集）

Set<T> operator-(const Set<T>& other) const {

Set<T> result;

Node<T>\* current = this->head;

while (current != nullptr) {

if (!other.contains(current->data)) {

result.add(current->data);

}

current = current->next;

}

return result;

}

// 集合的并（并集）

Set<T> operator+(const Set<T>& other) const {

Set<T> result = \*this; // 先复制当前集合

Node<T>\* current = other.head;

while (current != nullptr) {

if (!result.contains(current->data)) {

result.add(current->data);

}

current = current->next;

}

return result;

}

// 集合的交（交集）

Set<T> intersection(const Set<T>& other) const {

Set<T> result;

Node<T>\* current = this->head;

while (current != nullptr) {

if (other.contains(current->data)) {

result.add(current->data);

}

current = current->next;

}

return result;

}

// 检查当前集合是否包含特定元素

bool contains(const T& value) const {

Node<T>\* current = this->head;

while (current != nullptr) {

if (current->data == value) return true;

current = current->next;

}

return false;

}

// 添加元素到集合

void add(const T& value) {

if (!contains(value)) { // 确保集合中不包含重复元素

Node<T>\* new\_node = new Node<T>(value);

if (this->tail != nullptr) {

this->tail->next = new\_node;

new\_node->prev = this->tail;

} else {

this->head = new\_node;

}

this->tail = new\_node;

}

}

};

集合的差 (- 运算符重载): 返回一个新的集合，包含存在于当前集合但不在另一个集合中的元素。

集合的并 (+ 运算符重载): 返回一个新的集合，包含两个集合的所有唯一元素。

集合的交 (intersection 方法): 返回一个新的集合，包含两个集合中共同的元素。

使用示例

int main() {

Set<int> setA;

setA.add(1);

setA.add(2);

setA.add(3);

Set<int> setB;

setB.add(2);

setB.add(3);

setB.add(4);

Set<int> resultDiff = setA - setB;

Set<int> resultUnion = setA + setB;

Set<int> resultIntersection = setA.intersection(setB);

// 输出结果

return 0;

}