Q1:

Address.h:

```
    #ifndef ADDRESS_H
#define ADDRESS_H
#include <string>
#include "Person.cpp"
#pragma once
using namespace std;
class address
{
public:
    address(int block, int unit, int floor, string street, string city,
string country, int postalCode, Person person);
    friend std::ostream& operator<<(std::ostream& os, const address&
address);
    int getBlock();
    void setBlock(int block);

    int getUnit();
    void setUnit(int unit);

    int getFloor();
    void setFloor(int floor);

    string getStreet();
    void setStreet(string street);

    string getCity();
    void setCity(string city);

    string getCountry();
    void setCountry(string country);

    int getPostalCode();
    void setPostalCode(int postalCode);

    string getPerson();
    void setPerson(Person person);
    ~address();

private:
    int block;
    int unit;
    int floor;
```

```
    string street;
    string city;
    string country;
    int postalCode;
    Person person;
};

#endif
```

Address.cpp:

```
    #include "address.h"

address::address(int block, int unit, int floor, string street, string city,
string country, int postalCode, Person person)
{
    this->block = block;
    this->floor = floor;
    this->postalCode = postalCode;
    this->country = country;
    this->city = city;
    this->street = street;
    this->unit = unit;
    this->person = person;
}
std::ostream& operator<<(std::ostream& os, address& address) {
    os << address.getBlock() << "-" << address.getUnit() << "-" <<
address.getFloor() << ", "
       << address.getStreet() << ", " << address.getCity() << ", " <<
address.getCountry() << ", "
       << address.getPostalCode() << ", " << address.getPerson();
    return os;
}
int address::getBlock() { return block; }
void address::setBlock(int block) { this->block = block; }

int address::getUnit() { return unit; }
void address::setUnit(int unit) { this->unit = unit; }

int address::getFloor() { return floor; }
void address::setFloor(int floor) { this->floor = floor; }

string address::getStreet(){ return street; }
void address::setStreet(std::string street) { this->street = street; }
```

```
string address::getCity() { return city; }
void address::setCity(std::string city) { this->city = city; }

string address::getCountry() { return country; }
void address::setCountry(std::string country) { this->country = country; }

int address::getPostalCode() { return postalCode; }
void address::setPostalCode(int postalCode) { this->postalCode =
postalCode; }

string address::getPerson(){
    return (this->person).getFirstName() + " " +
(this->person).getMiddleName() + " " + (this->person).getLastName();
};
void address::setPerson(Person person){
    this->person = person;
}
address::~address()
{

}
```

Main:

```
    #include <string>
#include <iostream>
#include <vector>
#include "address.cpp"
using namespace std;
int main() {
    vector<address> addressBook;


    Person person1("John", "A", "Doe");
    address address1(123, 456, 7, "Main Street", "New York", "USA", 10001,
person1);
    addressBook.push_back(address1);

    Person person2("Jane", "B", "Smith");
    address address2(456, 789, 10, "Park Avenue", "Los Angeles", "USA",
90001, person2);

    addressBook.push_back(address2);

    for (auto& a : addressBook) {
```

```
        cout << a << endl;
    }

    return 0;
}
```

Res:

```
123-456-7, Main Street, New York, USA, 10001, John A Doe
456-789-10, Park Avenue, Los Angeles, USA, 90001, Jane B Smith
```

Q2:

```
#include <iostream>
#include <set>
using namespace std;
int main() {
    set<int> set1;
    set1.insert(1);
    set1.insert(2);
    set1.insert(3);
    set1.insert(4);
    set1.insert(5);
    set<int> set2;
    set2.insert(4);
    set2.insert(5);
    set2.insert(6);
    set2.insert(7);
    set2.insert(8);
    // Union
    set<int> unionSet;
    set_union(set1.begin(), set1.end(), set2.begin(), set2.end(),
inserter(unionSet, unionSet.begin()));
    cout << "Union of sets: ";
    for (auto it = unionSet.begin(); it != unionSet.end(); it++) {
        cout << *it << " ";
    }
    cout << ::endl;

    // Intersection
    set<int> intersectSet;
    set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(),
inserter(intersectSet, intersectSet.begin()));
    cout << "Intersection of sets: ";
```

```
    for (auto it = intersectSet.begin(); it != intersectSet.end(); it++) {
        cout << *it << " ";
    }
    cout << endl;

    // Difference
    set<int> diffSet;
    set_difference(set2.begin(), set2.end(), set1.begin(), set1.end(),
inserter(diffSet, diffSet.begin()));
    cout << "Difference of sets: ";
    for (auto it = diffSet.begin(); it != diffSet.end(); it++) {
        cout << *it << " ";
    }
    cout << endl;

    return 0;
}
```

Res:

```
Union of sets: 1 2 3 4 5 6 7 8
Intersection of sets: 4 5
Difference of sets: 6 7 8
```