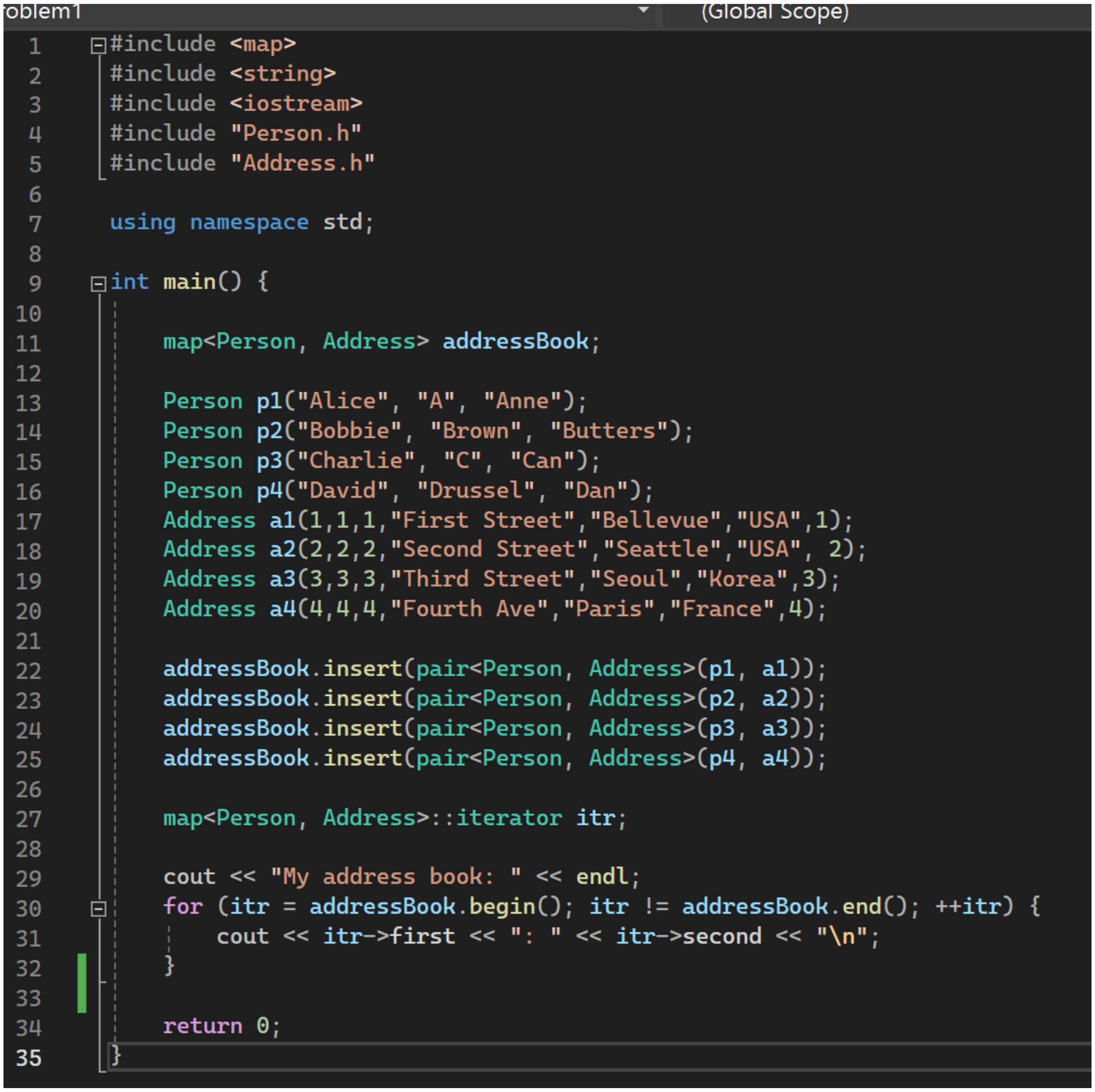
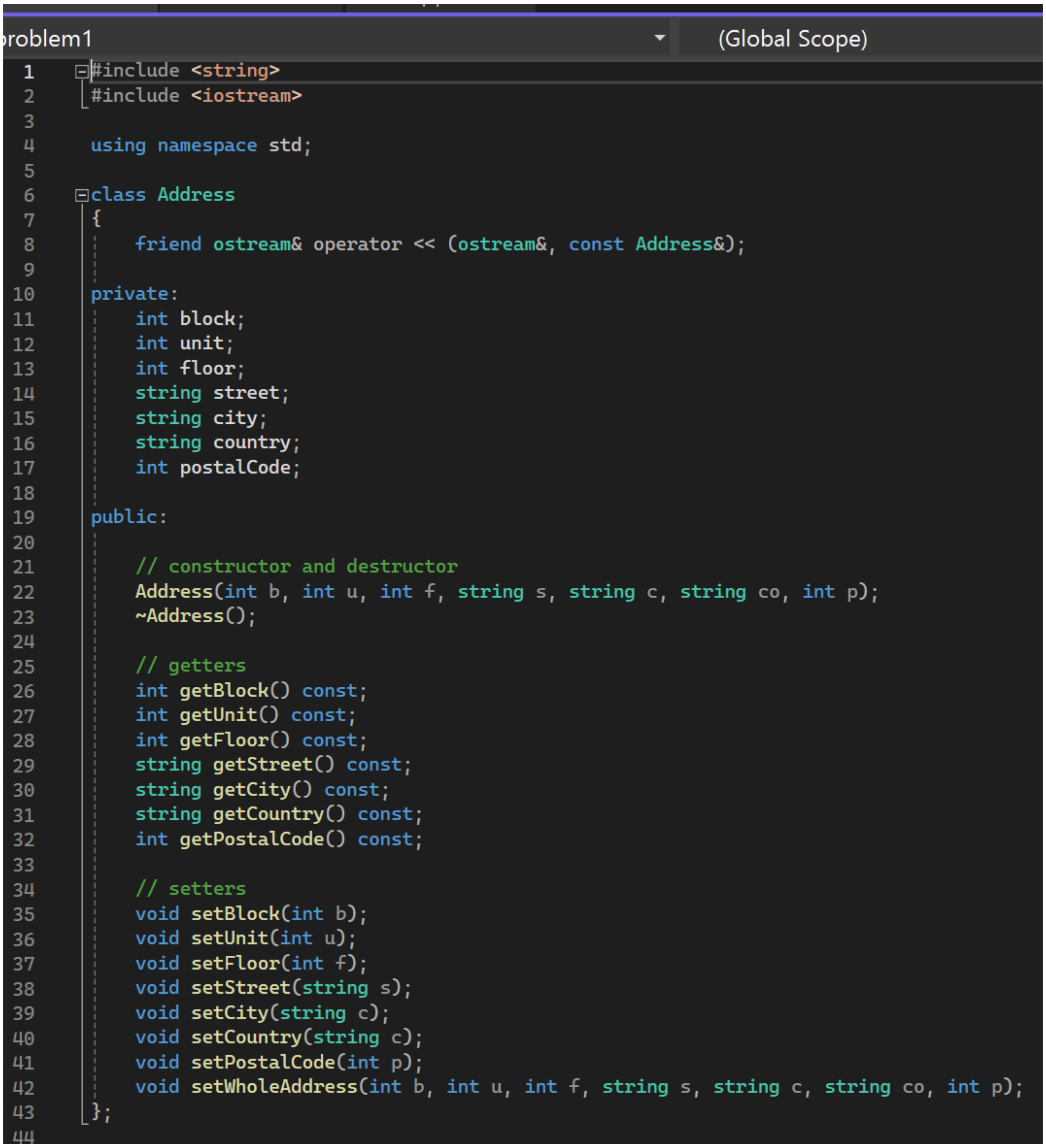
Problem 1
Code:
main.cpp


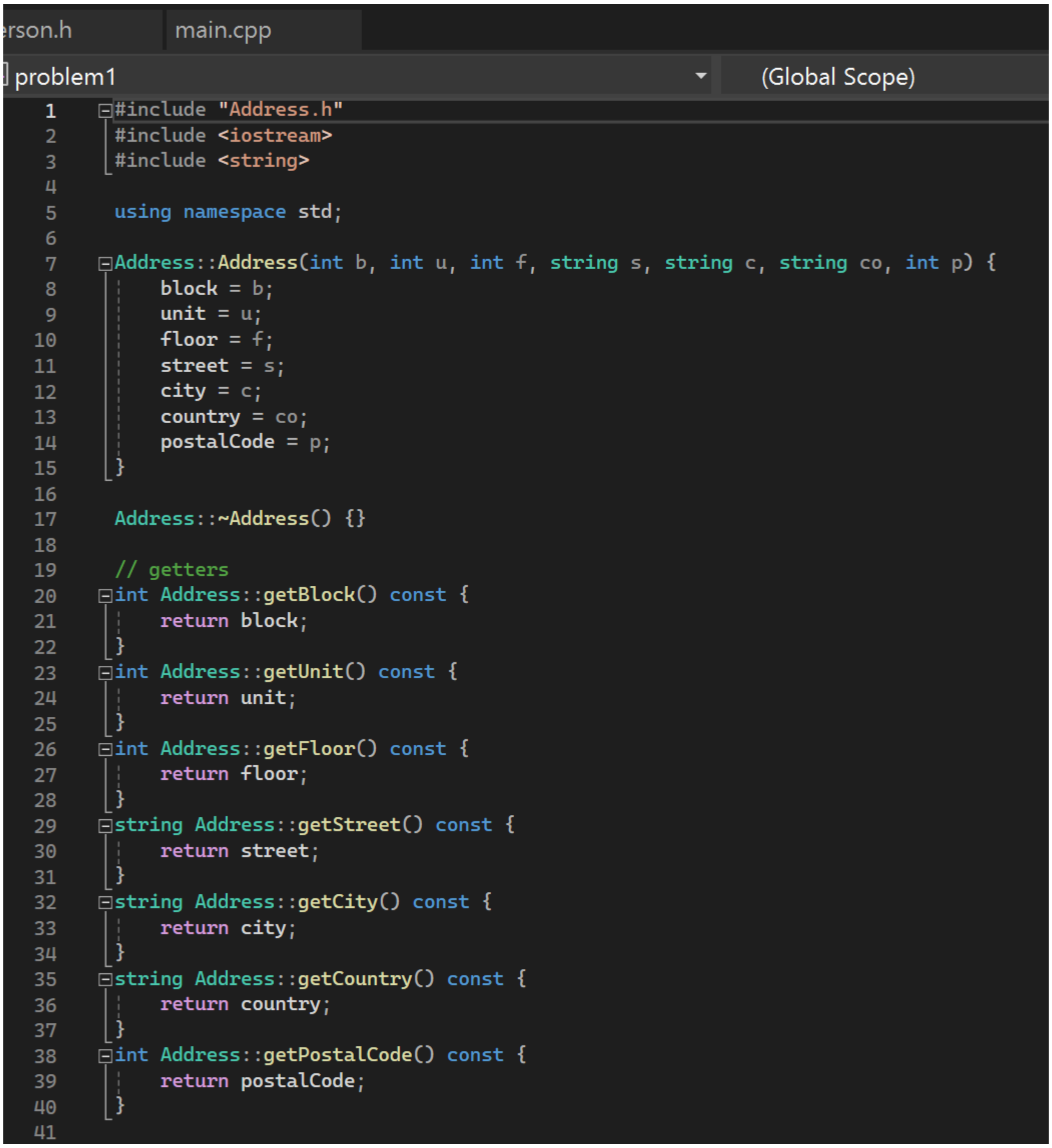
```cpp
#include <map>
#include <string>
#include <iostream>
#include "Person.h"
#include "Address.h"

using namespace std;

int main() {

    map<Person, Address> addressBook;

    Person p1("Alice", "A", "Anne");
    Person p2("Bobbie", "Brown", "Butters");
    Person p3("Charlie", "C", "Can");
    Person p4("David", "Drussel", "Dan");
    Address a1(1,1,1,"First Street","Bellevue","USA",1);
    Address a2(2,2,2,"Second Street","Seattle","USA", 2);
    Address a3(3,3,3,"Third Street","Seoul","Korea",3);
    Address a4(4,4,4,"Fourth Ave","Paris","France",4);

    addressBook.insert(pair<Person, Address>(p1, a1));
    addressBook.insert(pair<Person, Address>(p2, a2));
    addressBook.insert(pair<Person, Address>(p3, a3));
    addressBook.insert(pair<Person, Address>(p4, a4));

    map<Person, Address>::iterator itr;

    cout << "My address book: " << endl;
    for (itr = addressBook.begin(); itr != addressBook.end(); ++itr) {
        cout << itr->first << ": " << itr->second << "\n";
    }

    return 0;
}
```

Address.h



```cpp
#include <string>
#include <iostream>

using namespace std;

class Address
{
    friend ostream& operator << (ostream&, const Address&);

private:
    int block;
    int unit;
    int floor;
    string street;
    string city;
    string country;
    int postalCode;

public:

    // constructor and destructor
    Address(int b, int u, int f, string s, string c, string co, int p);
    ~Address();

    // getters
    int getBlock() const;
    int getUnit() const;
    int getFloor() const;
    string getStreet() const;
    string getCity() const;
    string getCountry() const;
    int getPostalCode() const;

    // setters
    void setBlock(int b);
    void setUnit(int u);
    void setFloor(int f);
    void setStreet(string s);
    void setCity(string c);
    void setCountry(string c);
    void setPostalCode(int p);
    void setWholeAddress(int b, int u, int f, string s, string c, string co, int p);
};
```

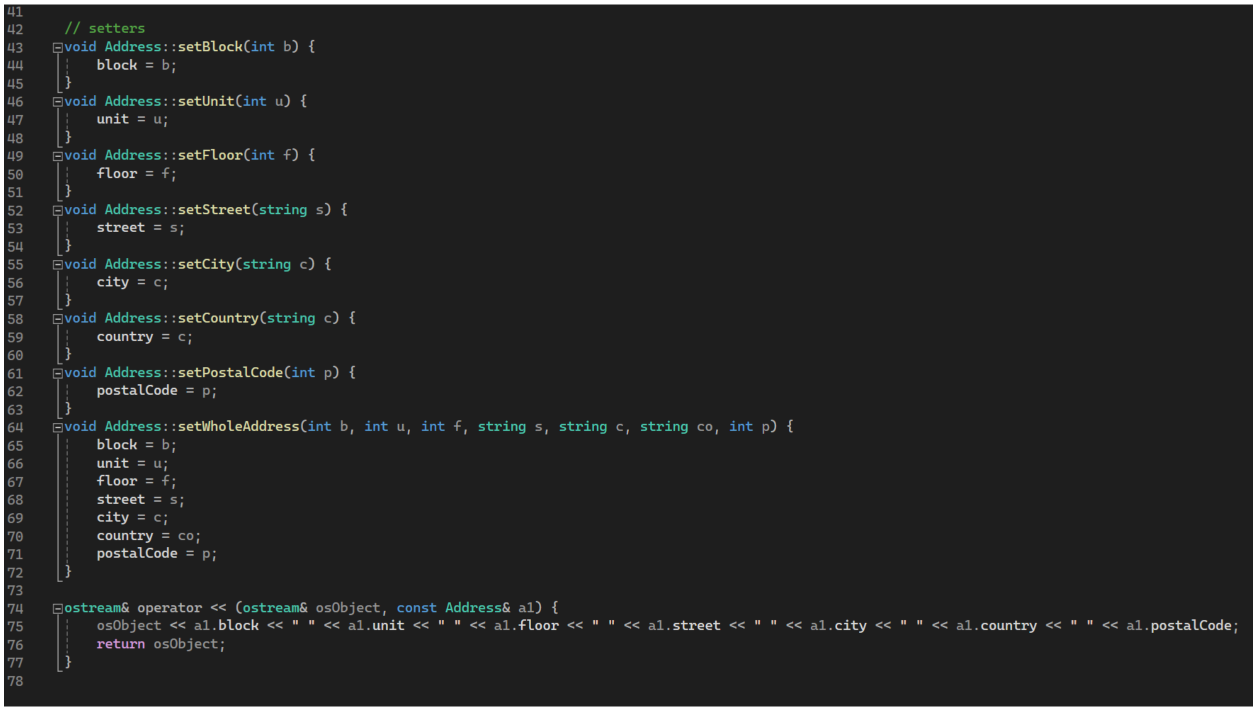
address.cpp



rson.h   main.cpp

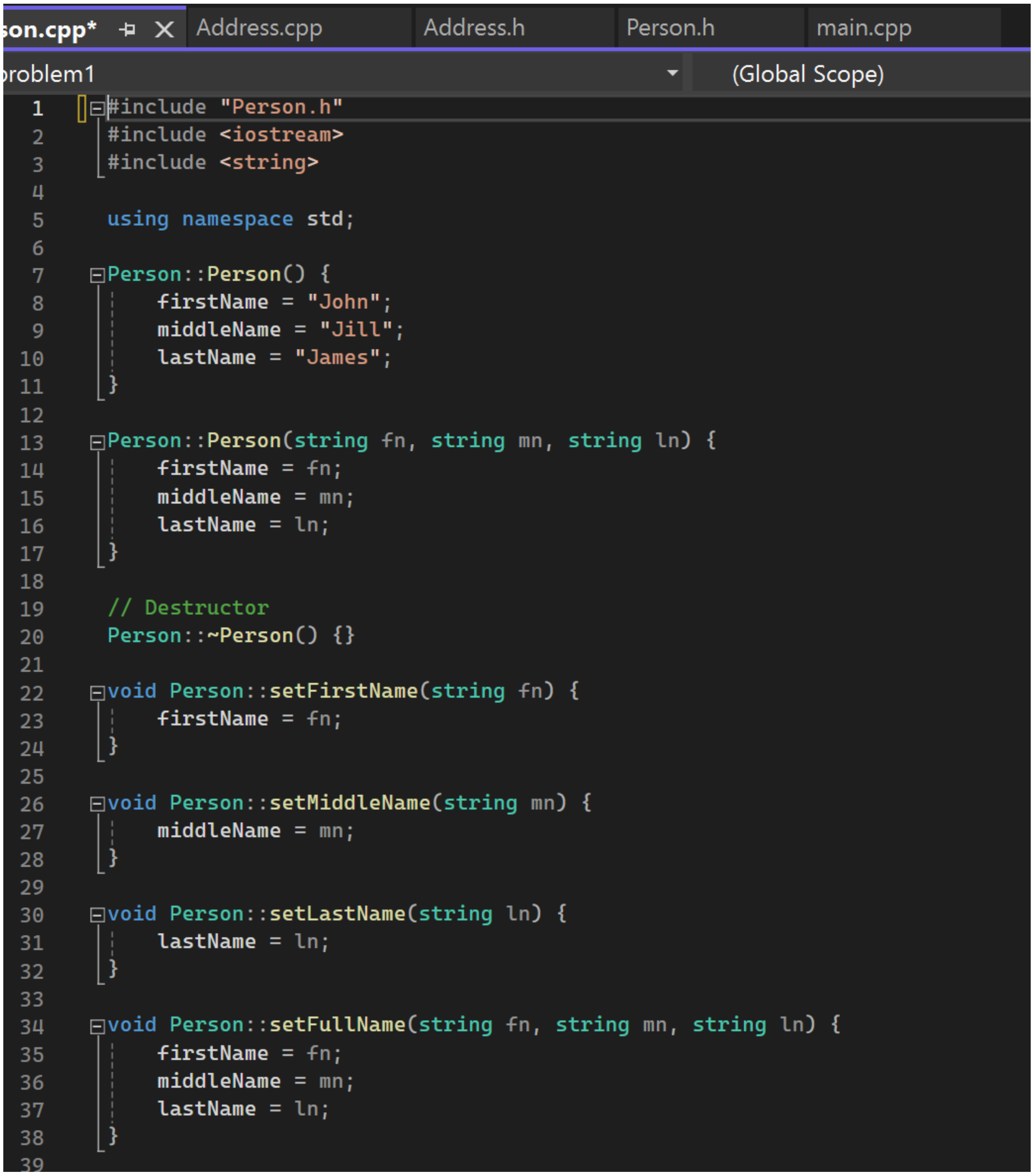
problem1   (Global Scope)

```cpp
#include "Address.h"
#include <iostream>
#include <string>

using namespace std;

Address::Address(int b, int u, int f, string s, string c, string co, int p) {
    block = b;
    unit = u;
    floor = f;
    street = s;
    city = c;
    country = co;
    postalCode = p;
}

Address::~Address() {}

// getters
int Address::getBlock() const {
    return block;
}
int Address::getUnit() const {
    return unit;
}
int Address::getFloor() const {
    return floor;
}
string Address::getStreet() const {
    return street;
}
string Address::getCity() const {
    return city;
}
string Address::getCountry() const {
    return country;
}
int Address::getPostalCode() const {
    return postalCode;
}

```

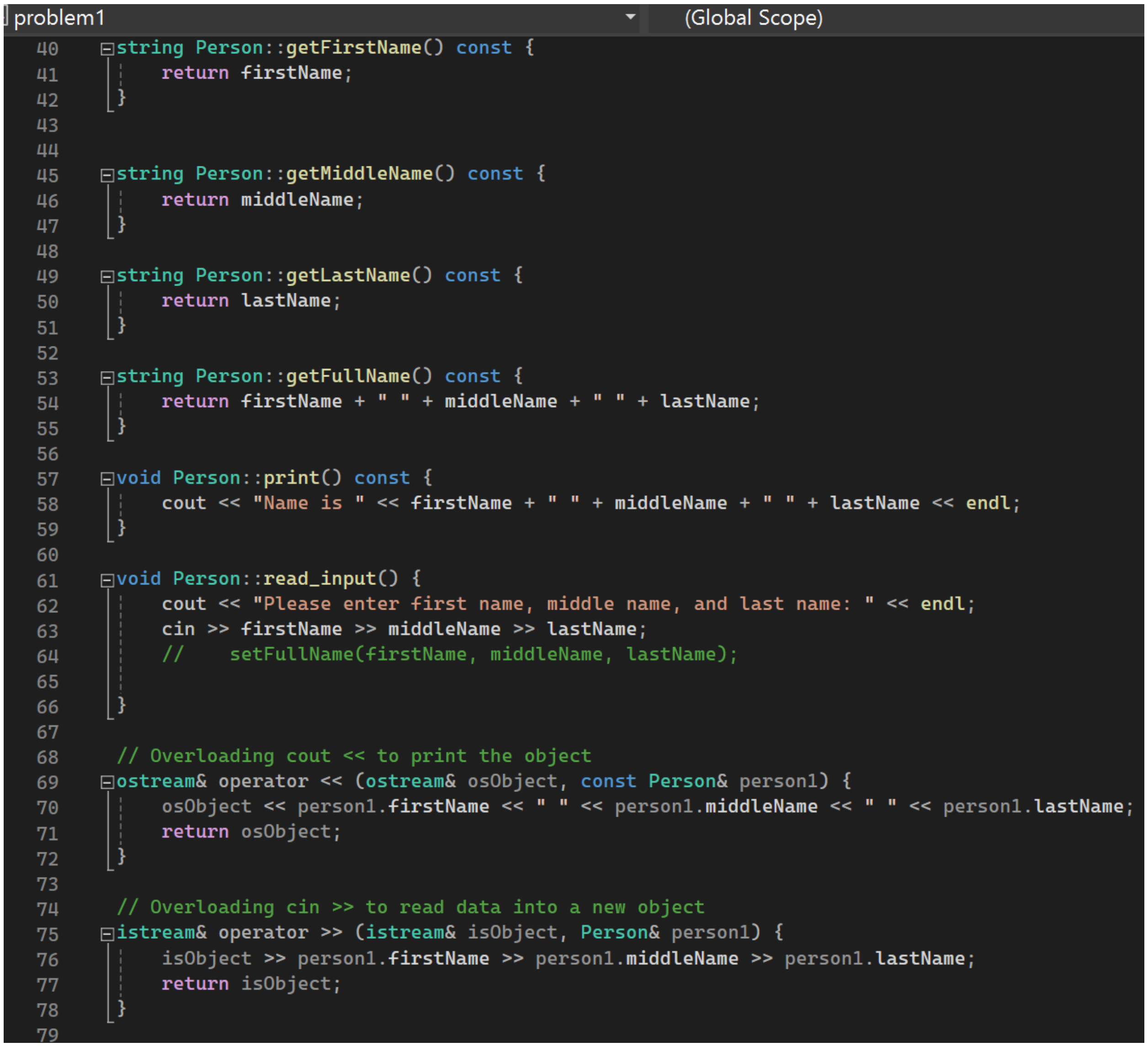


```cpp

// setters
void Address::setBlock(int b) {
    block = b;
}
void Address::setUnit(int u) {
    unit = u;
}
void Address::setFloor(int f) {
    floor = f;
}
void Address::setStreet(string s) {
    street = s;
}
void Address::setCity(string c) {
    city = c;
}
void Address::setCountry(string c) {
    country = c;
}
void Address::setPostalCode(int p) {
    postalCode = p;
}
void Address::setWholeAddress(int b, int u, int f, string s, string c, string co, int p) {
    block = b;
    unit = u;
    floor = f;
    street = s;
    city = c;
    country = co;
    postalCode = p;
}

ostream& operator << (ostream& osObject, const Address& a1) {
    osObject << a1.block << " " << a1.unit << " " << a1.floor << " " << a1.street << " " << a1.city << " " << a1.country << " " << a1.postalCode;
    return osObject;
}

```

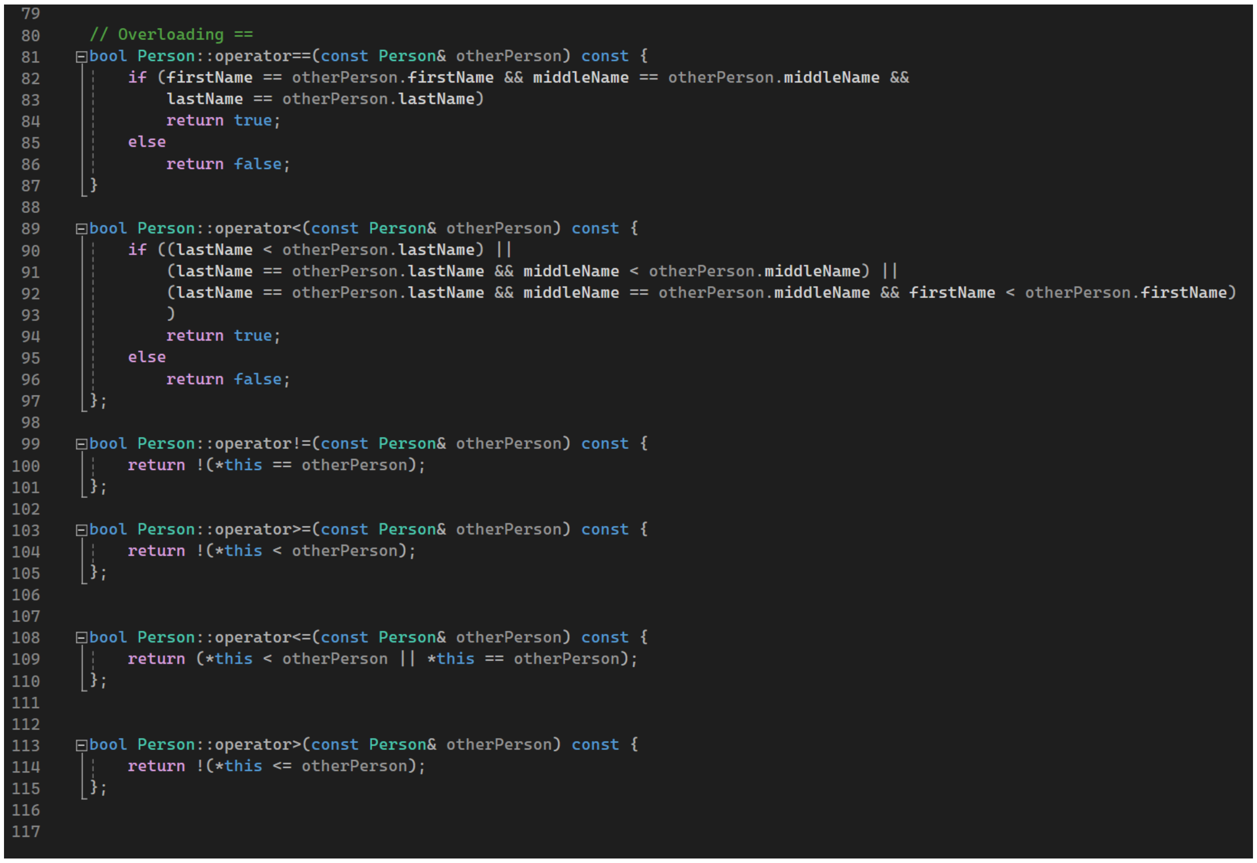
Person.cpp



```cpp
#include "Person.h"
#include <iostream>
#include <string>

using namespace std;

Person::Person() {
    firstName = "John";
    middleName = "Jill";
    lastName = "James";
}

Person::Person(string fn, string mn, string ln) {
    firstName = fn;
    middleName = mn;
    lastName = ln;
}

// Destructor
Person::~Person() {}

void Person::setFirstName(string fn) {
    firstName = fn;
}

void Person::setMiddleName(string mn) {
    middleName = mn;
}

void Person::setLastName(string ln) {
    lastName = ln;
}

void Person::setFullName(string fn, string mn, string ln) {
    firstName = fn;
    middleName = mn;
    lastName = ln;
}

```

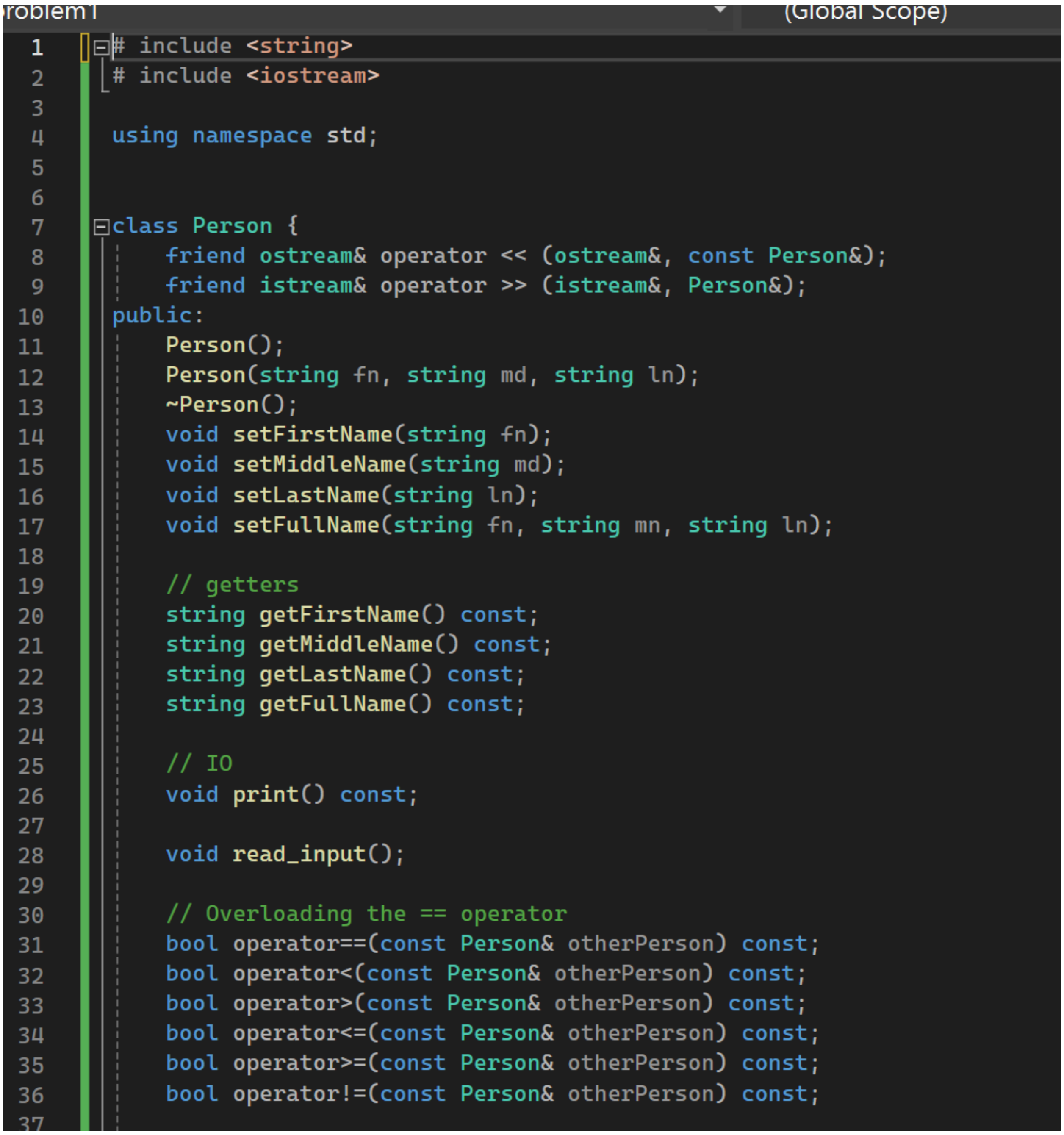


```cpp
string Person::getFirstName() const {
    return firstName;
}


string Person::getMiddleName() const {
    return middleName;
}

string Person::getLastName() const {
    return lastName;
}

string Person::getFullName() const {
    return firstName + " " + middleName + " " + lastName;
}

void Person::print() const {
    cout << "Name is " << firstName + " " + middleName + " " + lastName << endl;
}

void Person::read_input() {
    cout << "Please enter first name, middle name, and last name: " << endl;
    cin >> firstName >> middleName >> lastName;
    //    setFullName(firstName, middleName, lastName);

}

// Overloading cout << to print the object
ostream& operator << (ostream& osObject, const Person& person1) {
    osObject << person1.firstName << " " << person1.middleName << " " << person1.lastName;
    return osObject;
}

// Overloading cin >> to read data into a new object
istream& operator >> (istream& isObject, Person& person1) {
    isObject >> person1.firstName >> person1.middleName >> person1.lastName;
    return isObject;
}

```

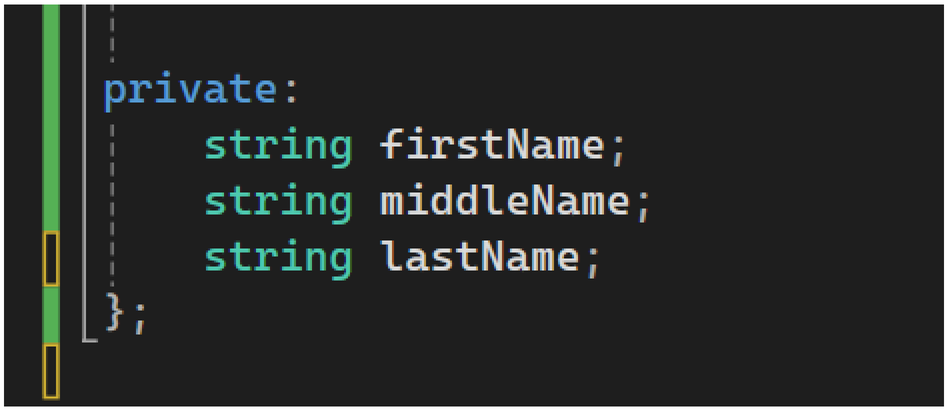


```cpp

// Overloading ==
bool Person::operator==(const Person& otherPerson) const {
    if (firstName == otherPerson.firstName && middleName == otherPerson.middleName &&
        lastName == otherPerson.lastName)
        return true;
    else
        return false;
}

bool Person::operator<(const Person& otherPerson) const {
    if ((lastName < otherPerson.lastName) ||
        (lastName == otherPerson.lastName && middleName < otherPerson.middleName) ||
        (lastName == otherPerson.lastName && middleName == otherPerson.middleName && firstName < otherPerson.firstName)
        )
        return true;
    else
        return false;
};

bool Person::operator!=(const Person& otherPerson) const {
    return !(*this == otherPerson);
};

bool Person::operator>=(const Person& otherPerson) const {
    return !(*this < otherPerson);
};


bool Person::operator<=(const Person& otherPerson) const {
    return (*this < otherPerson || *this == otherPerson);
};


bool Person::operator>(const Person& otherPerson) const {
    return !(*this <= otherPerson);
};


```

Person.h

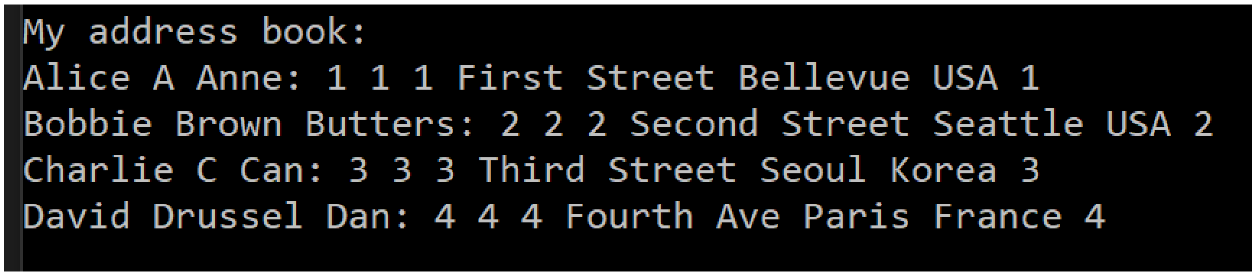


```cpp
# include <string>
# include <iostream>

using namespace std;


class Person {
    friend ostream& operator << (ostream&, const Person&);
    friend istream& operator >> (istream&, Person&);
public:
    Person();
    Person(string fn, string md, string ln);
    ~Person();
    void setFirstName(string fn);
    void setMiddleName(string md);
    void setLastName(string ln);
    void setFullName(string fn, string mn, string ln);

    // getters
    string getFirstName() const;
    string getMiddleName() const;
    string getLastName() const;
    string getFullName() const;

    // IO
    void print() const;

    void read_input();

    // Overloading the == operator
    bool operator==(const Person& otherPerson) const;
    bool operator<(const Person& otherPerson) const;
    bool operator>(const Person& otherPerson) const;
    bool operator<=(const Person& otherPerson) const;
    bool operator>=(const Person& otherPerson) const;
    bool operator!=(const Person& otherPerson) const;
```



```
private:
    string firstName;
    string middleName;
    string lastName;
};
```

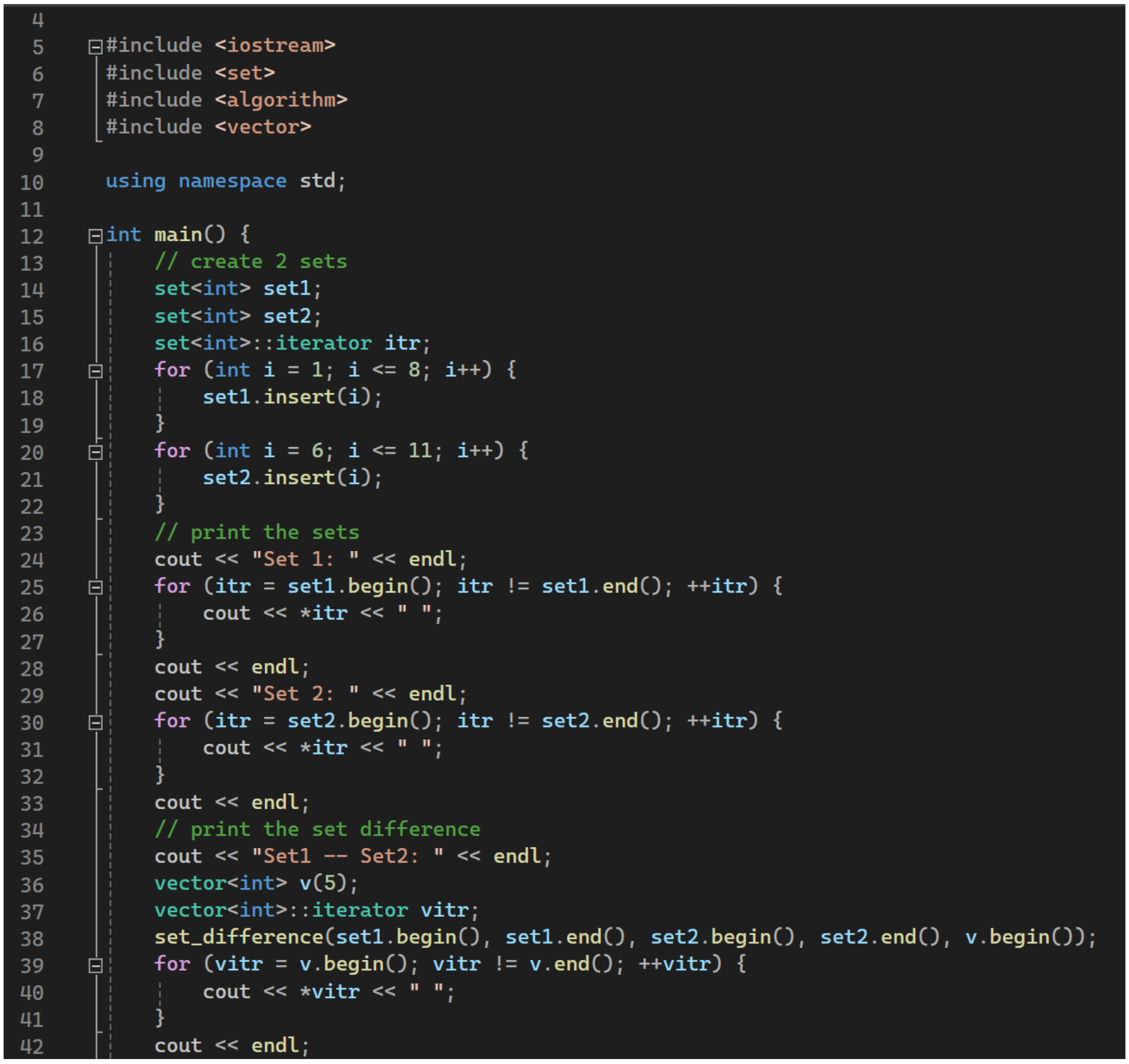
Output:



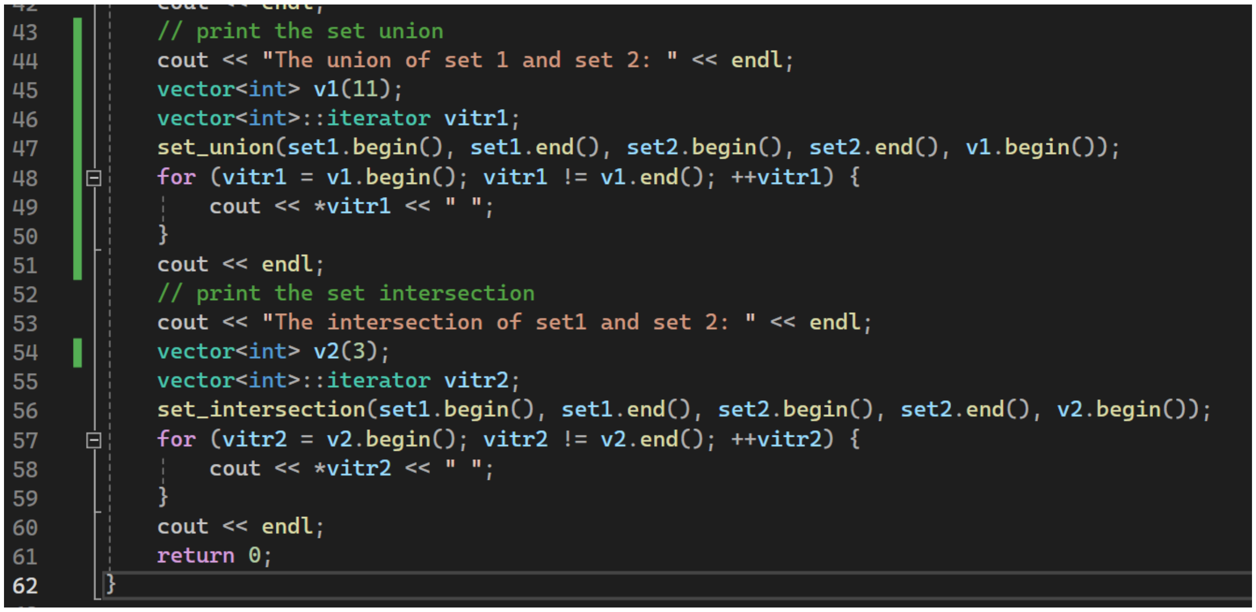
```
My address book:
Alice A Anne: 1 1 1 First Street Bellevue USA 1
Bobbie Brown Butters: 2 2 2 Second Street Seattle USA 2
Charlie C Can: 3 3 3 Third Street Seoul Korea 3
David Drussel Dan: 4 4 4 Fourth Ave Paris France 4
```

Problem 2

Code:

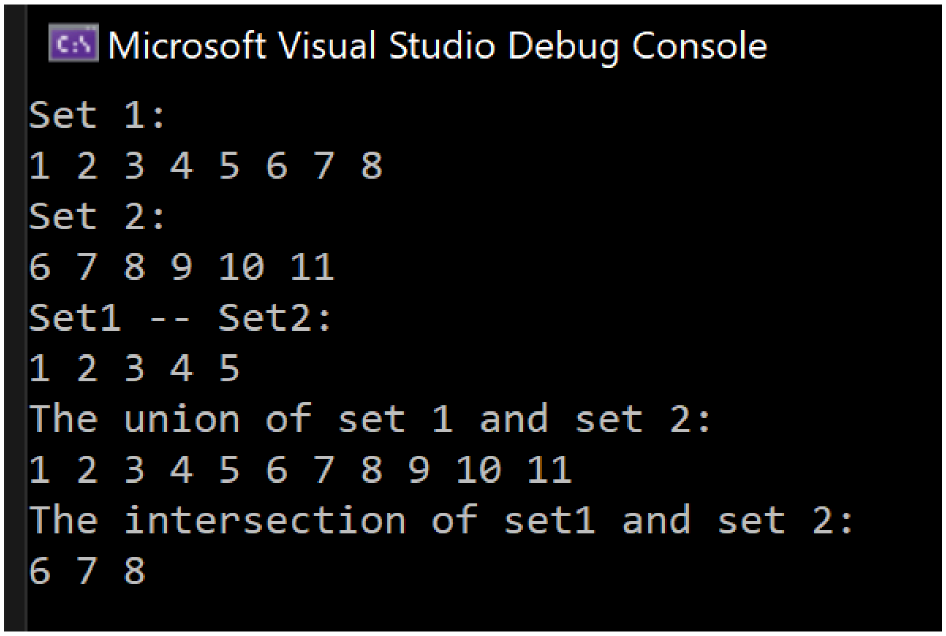


```cpp

#include <iostream>
#include <set>
#include <algorithm>
#include <vector>

using namespace std;

int main() {
    // create 2 sets
    set<int> set1;
    set<int> set2;
    set<int>::iterator itr;
    for (int i = 1; i <= 8; i++) {
        set1.insert(i);
    }
    for (int i = 6; i <= 11; i++) {
        set2.insert(i);
    }
    // print the sets
    cout << "Set 1: " << endl;
    for (itr = set1.begin(); itr != set1.end(); ++itr) {
        cout << *itr << " ";
    }
    cout << endl;
    cout << "Set 2: " << endl;
    for (itr = set2.begin(); itr != set2.end(); ++itr) {
        cout << *itr << " ";
    }
    cout << endl;
    // print the set difference
    cout << "Set1 -- Set2: " << endl;
    vector<int> v(5);
    vector<int>::iterator vitr;
    set_difference(set1.begin(), set1.end(), set2.begin(), set2.end(), v.begin());
    for (vitr = v.begin(); vitr != v.end(); ++vitr) {
        cout << *vitr << " ";
    }
    cout << endl;
```



```cpp
    cout << endl;
    // print the set union
    cout << "The union of set 1 and set 2: " << endl;
    vector<int> v1(11);
    vector<int>::iterator vitr1;
    set_union(set1.begin(), set1.end(), set2.begin(), set2.end(), v1.begin());
    for (vitr1 = v1.begin(); vitr1 != v1.end(); ++vitr1) {
        cout << *vitr1 << " ";
    }
    cout << endl;
    // print the set intersection
    cout << "The intersection of set1 and set 2: " << endl;
    vector<int> v2(3);
    vector<int>::iterator vitr2;
    set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), v2.begin());
    for (vitr2 = v2.begin(); vitr2 != v2.end(); ++vitr2) {
        cout << *vitr2 << " ";
    }
    cout << endl;
    return 0;
}
```

Output:



Microsoft Visual Studio Debug Console

```
Set 1:
1 2 3 4 5 6 7 8
Set 2:
6 7 8 9 10 11
Set1 -- Set2:
1 2 3 4 5
The union of set 1 and set 2:
1 2 3 4 5 6 7 8 9 10 11
The intersection of set1 and set 2:
6 7 8
```