

Problem 1

Code:

main.cpp

```
Problem1 (Global Scope)
1  #include <map>
2  #include <string>
3  #include <iostream>
4  #include "Person.h"
5  #include "Address.h"
6
7  using namespace std;
8
9  int main() {
10
11     map<Person, Address> addressBook;
12
13     Person p1("Alice", "A", "Anne");
14     Person p2("Bobbie", "Brown", "Butters");
15     Person p3("Charlie", "C", "Can");
16     Person p4("David", "Drussel", "Dan");
17     Address a1(1,1,1,"First Street","Bellevue","USA",1);
18     Address a2(2,2,2,"Second Street","Seattle","USA", 2);
19     Address a3(3,3,3,"Third Street","Seoul","Korea",3);
20     Address a4(4,4,4,"Fourth Ave","Paris","France",4);
21
22     addressBook.insert(pair<Person, Address>(p1, a1));
23     addressBook.insert(pair<Person, Address>(p2, a2));
24     addressBook.insert(pair<Person, Address>(p3, a3));
25     addressBook.insert(pair<Person, Address>(p4, a4));
26
27     map<Person, Address>::iterator itr;
28
29     cout << "My address book: " << endl;
30     for (itr = addressBook.begin(); itr != addressBook.end(); ++itr) {
31         cout << itr->first << ": " << itr->second << "\n";
32     }
33
34     return 0;
35 }
```

Address.h

problem1

(Global Scope)

```
1  #include <string>
2  #include <iostream>
3
4  using namespace std;
5
6  class Address
7  {
8      friend ostream& operator << (ostream&, const Address&);
9
10     private:
11         int block;
12         int unit;
13         int floor;
14         string street;
15         string city;
16         string country;
17         int postalCode;
18
19     public:
20
21         // constructor and destructor
22         Address(int b, int u, int f, string s, string c, string co, int p);
23         ~Address();
24
25         // getters
26         int getBlock() const;
27         int getUnit() const;
28         int getFloor() const;
29         string getStreet() const;
30         string getCity() const;
31         string getCountry() const;
32         int getPostalCode() const;
33
34         // setters
35         void setBlock(int b);
36         void setUnit(int u);
37         void setFloor(int f);
38         void setStreet(string s);
39         void setCity(string c);
40         void setCountry(string c);
41         void setPostalCode(int p);
42         void setWholeAddress(int b, int u, int f, string s, string c, string co, int p);
43     };
44
```

address.cpp

person.h

main.cpp

problem1

(Global Scope)

```
1  #include "Address.h"
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6
7  Address::Address(int b, int u, int f, string s, string c, string co, int p) {
8      block = b;
9      unit = u;
10     floor = f;
11     street = s;
12     city = c;
13     country = co;
14     postalCode = p;
15 }
16
17 Address::~~Address() {}
18
19 // getters
20 int Address::getBlock() const {
21     return block;
22 }
23 int Address::getUnit() const {
24     return unit;
25 }
26 int Address::getFloor() const {
27     return floor;
28 }
29 string Address::getStreet() const {
30     return street;
31 }
32 string Address::getCity() const {
33     return city;
34 }
35 string Address::getCountry() const {
36     return country;
37 }
38 int Address::getPostalCode() const {
39     return postalCode;
40 }
41
```

```

41
42 // setters
43 void Address::setBlock(int b) {
44     block = b;
45 }
46 void Address::setUnit(int u) {
47     unit = u;
48 }
49 void Address::setFloor(int f) {
50     floor = f;
51 }
52 void Address::setStreet(string s) {
53     street = s;
54 }
55 void Address::setCity(string c) {
56     city = c;
57 }
58 void Address::setCountry(string c) {
59     country = c;
60 }
61 void Address::setPostalCode(int p) {
62     postalCode = p;
63 }
64 void Address::setWholeAddress(int b, int u, int f, string s, string c, string co, int p) {
65     block = b;
66     unit = u;
67     floor = f;
68     street = s;
69     city = c;
70     country = co;
71     postalCode = p;
72 }
73
74 ostream& operator << (ostream& osObject, const Address& a1) {
75     osObject << a1.block << " " << a1.unit << " " << a1.floor << " " << a1.street << " " << a1.city << " " << a1.country << " " << a1.postalCode;
76     return osObject;
77 }
78

```

Person.cpp

```
son.cpp*  Address.cpp  Address.h  Person.h  main.cpp
problem1  (Global Scope)
1  #include "Person.h"
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6
7  Person::Person() {
8      firstName = "John";
9      middleName = "Jill";
10     lastName = "James";
11 }
12
13 Person::Person(string fn, string mn, string ln) {
14     firstName = fn;
15     middleName = mn;
16     lastName = ln;
17 }
18
19 // Destructor
20 Person::~~Person() {}
21
22 void Person::setFirstName(string fn) {
23     firstName = fn;
24 }
25
26 void Person::setMiddleName(string mn) {
27     middleName = mn;
28 }
29
30 void Person::setLastName(string ln) {
31     lastName = ln;
32 }
33
34 void Person::setFullName(string fn, string mn, string ln) {
35     firstName = fn;
36     middleName = mn;
37     lastName = ln;
38 }
39
```

```
problem1 (Global Scope)
40  string Person::getFirstName() const {
41      return firstName;
42  }
43
44
45  string Person::getMiddleName() const {
46      return middleName;
47  }
48
49  string Person::getLastName() const {
50      return lastName;
51  }
52
53  string Person::getFullName() const {
54      return firstName + " " + middleName + " " + lastName;
55  }
56
57  void Person::print() const {
58      cout << "Name is " << firstName + " " + middleName + " " + lastName << endl;
59  }
60
61  void Person::read_input() {
62      cout << "Please enter first name, middle name, and last name: " << endl;
63      cin >> firstName >> middleName >> lastName;
64      // setFullName(firstName, middleName, lastName);
65  }
66
67
68  // Overloading cout << to print the object
69  ostream& operator << (ostream& osObject, const Person& person1) {
70      osObject << person1.firstName << " " << person1.middleName << " " << person1.lastName;
71      return osObject;
72  }
73
74  // Overloading cin >> to read data into a new object
75  istream& operator >> (istream& isObject, Person& person1) {
76      isObject >> person1.firstName >> person1.middleName >> person1.lastName;
77      return isObject;
78  }
79
```

```

79
80 // Overloading ==
81 bool Person::operator==(const Person& otherPerson) const {
82     if (firstName == otherPerson.firstName && middleName == otherPerson.middleName &&
83         lastName == otherPerson.lastName)
84         return true;
85     else
86         return false;
87 }
88
89 bool Person::operator<(const Person& otherPerson) const {
90     if ((lastName < otherPerson.lastName) ||
91         (lastName == otherPerson.lastName && middleName < otherPerson.middleName) ||
92         (lastName == otherPerson.lastName && middleName == otherPerson.middleName && firstName < otherPerson.firstName)
93     )
94         return true;
95     else
96         return false;
97 };
98
99 bool Person::operator!=(const Person& otherPerson) const {
100     return !(*this == otherPerson);
101 };
102
103 bool Person::operator>=(const Person& otherPerson) const {
104     return !(*this < otherPerson);
105 };
106
107
108 bool Person::operator<=(const Person& otherPerson) const {
109     return (*this < otherPerson || *this == otherPerson);
110 };
111
112
113 bool Person::operator>(const Person& otherPerson) const {
114     return !(*this <= otherPerson);
115 };
116
117

```

Person.h

problem1

(Global Scope)

```

1  #include <string>
2  #include <iostream>
3
4  using namespace std;
5
6
7  class Person {
8      friend ostream& operator << (ostream&, const Person&);
9      friend istream& operator >> (istream&, Person&);
10 public:
11     Person();
12     Person(string fn, string md, string ln);
13     ~Person();
14     void setFirstName(string fn);
15     void setMiddleName(string md);
16     void setLastName(string ln);
17     void setFullName(string fn, string mn, string ln);
18
19     // getters
20     string getFirstName() const;
21     string getMiddleName() const;
22     string getLastName() const;
23     string getFullName() const;
24
25     // IO
26     void print() const;
27
28     void read_input();
29
30     // Overloading the == operator
31     bool operator==(const Person& otherPerson) const;
32     bool operator<(const Person& otherPerson) const;
33     bool operator>(const Person& otherPerson) const;
34     bool operator<=(const Person& otherPerson) const;
35     bool operator>=(const Person& otherPerson) const;
36     bool operator!=(const Person& otherPerson) const;
37

```



```
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:

```

4
5 #include <iostream>
6 #include <set>
7 #include <algorithm>
8 #include <vector>
9
10 using namespace std;
11
12 int main() {
13     // create 2 sets
14     set<int> set1;
15     set<int> set2;
16     set<int>::iterator itr;
17     for (int i = 1; i <= 8; i++) {
18         set1.insert(i);
19     }
20     for (int i = 6; i <= 11; i++) {
21         set2.insert(i);
22     }
23     // print the sets
24     cout << "Set 1: " << endl;
25     for (itr = set1.begin(); itr != set1.end(); ++itr) {
26         cout << *itr << " ";
27     }
28     cout << endl;
29     cout << "Set 2: " << endl;
30     for (itr = set2.begin(); itr != set2.end(); ++itr) {
31         cout << *itr << " ";
32     }
33     cout << endl;
34     // print the set difference
35     cout << "Set1 -- Set2: " << endl;
36     vector<int> v(5);
37     vector<int>::iterator vitr;
38     set_difference(set1.begin(), set1.end(), set2.begin(), set2.end(), v.begin());
39     for (vitr = v.begin(); vitr != v.end(); ++vitr) {
40         cout << *vitr << " ";
41     }
42     cout << endl;

```

```

42     cout << endl;
43     // print the set union
44     cout << "The union of set 1 and set 2: " << endl;
45     vector<int> v1(11);
46     vector<int>::iterator vitr1;
47     set_union(set1.begin(), set1.end(), set2.begin(), set2.end(), v1.begin());
48     for (vitr1 = v1.begin(); vitr1 != v1.end(); ++vitr1) {
49         cout << *vitr1 << " ";
50     }
51     cout << endl;
52     // print the set intersection
53     cout << "The intersection of set1 and set 2: " << endl;
54     vector<int> v2(3);
55     vector<int>::iterator vitr2;
56     set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), v2.begin());
57     for (vitr2 = v2.begin(); vitr2 != v2.end(); ++vitr2) {
58         cout << *vitr2 << " ";
59     }
60     cout << endl;
61     return 0;
62 }

```

Output:

```

C:\> 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

```