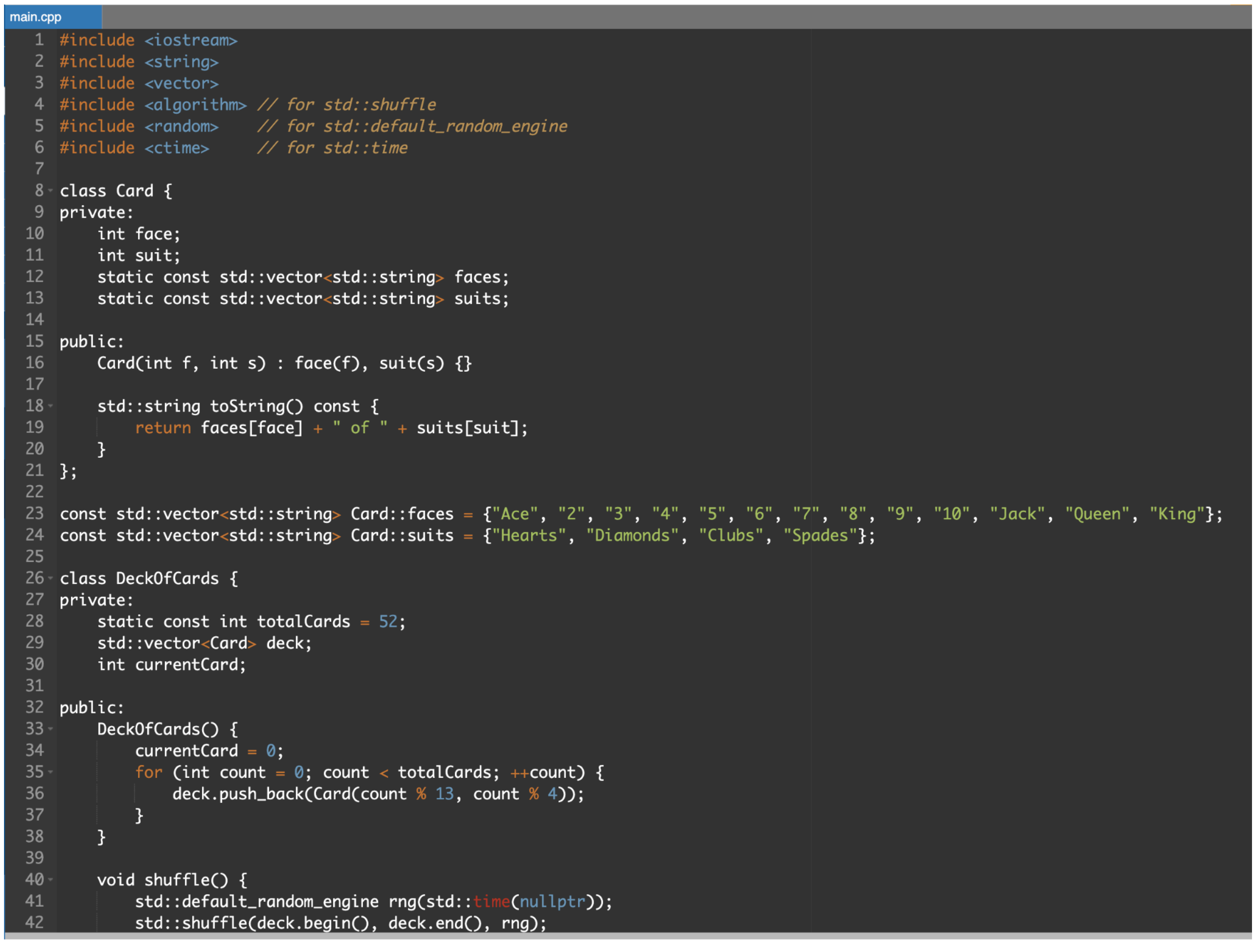
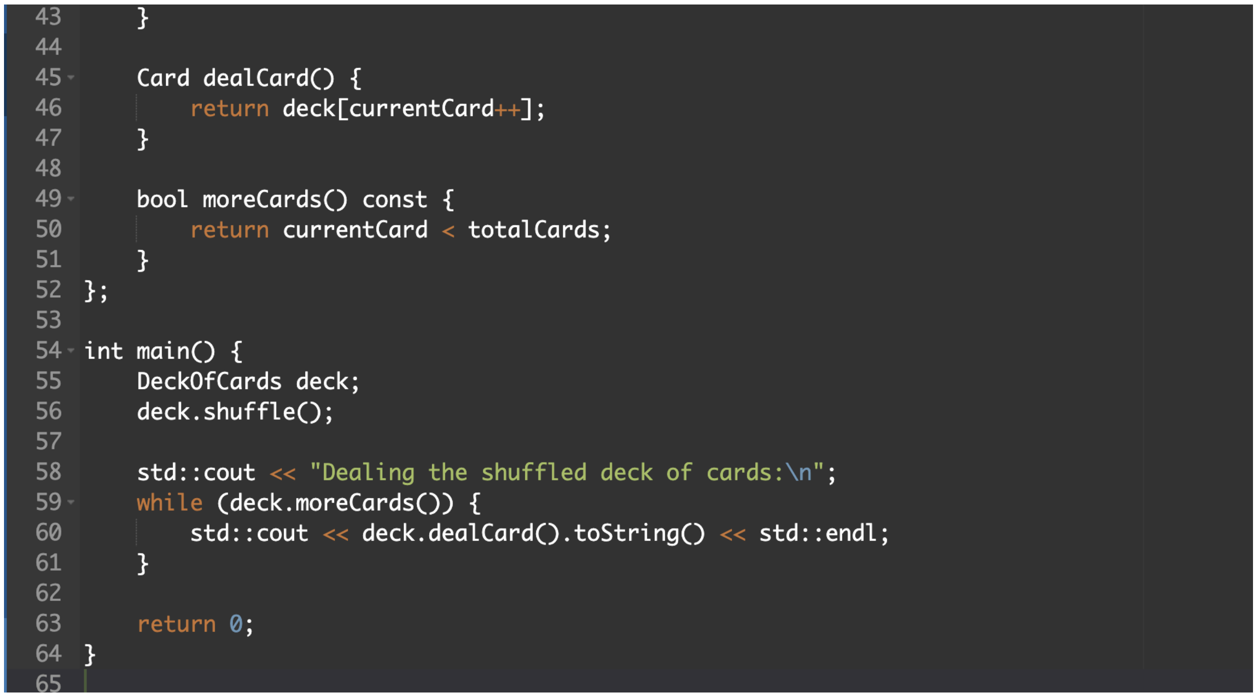
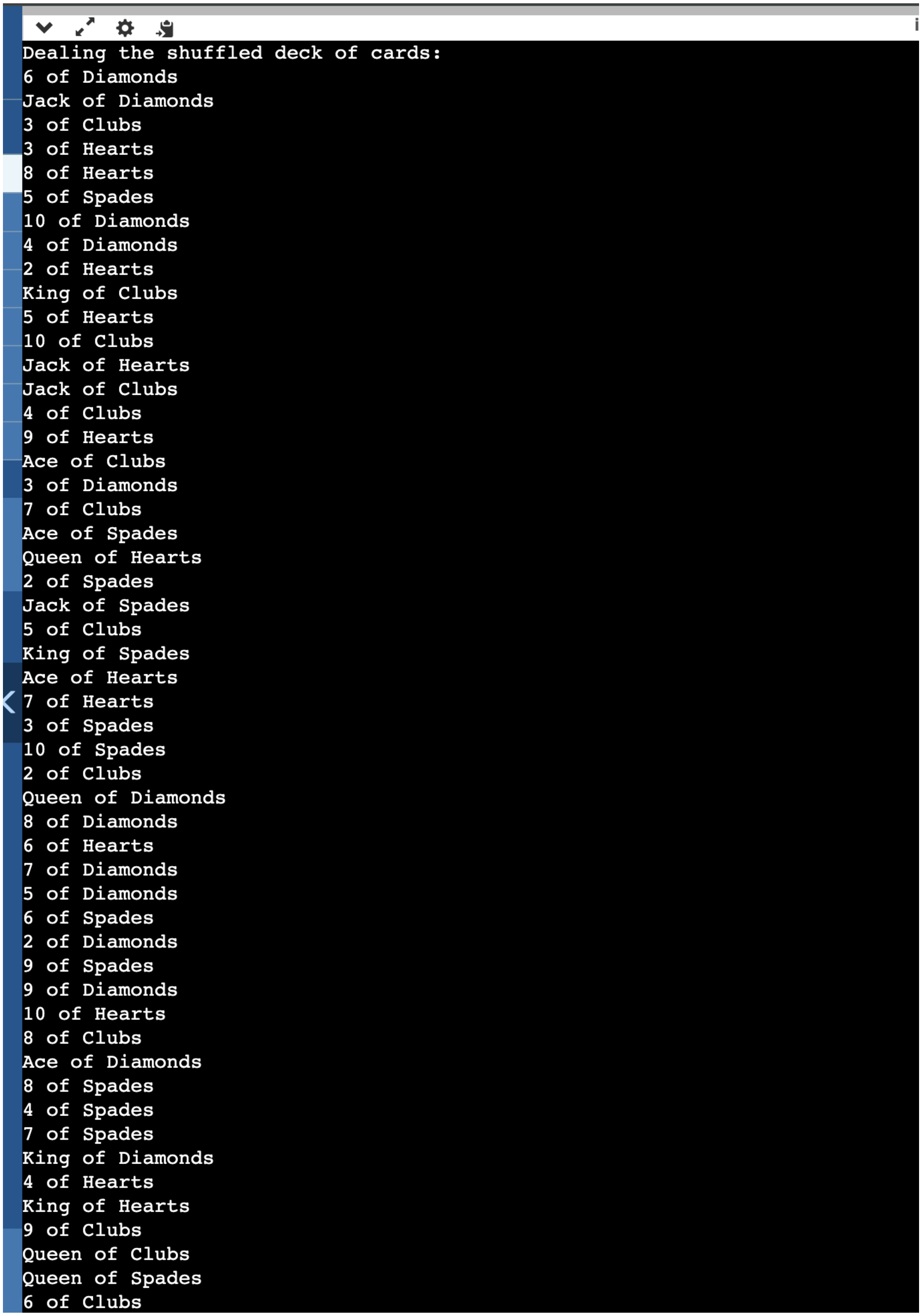
Samir Khadka
CS360
Question 1:


main.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm> // for std::shuffle
#include <random>    // for std::default_random_engine
#include <ctime>     // for std::time

class Card {
private:
    int face;
    int suit;
    static const std::vector<std::string> faces;
    static const std::vector<std::string> suits;

public:
    Card(int f, int s) : face(f), suit(s) {}

    std::string toString() const {
        return faces[face] + " of " + suits[suit];
    }
};

const std::vector<std::string> Card::faces = {"Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King"};
const std::vector<std::string> Card::suits = {"Hearts", "Diamonds", "Clubs", "Spades"};

class DeckOfCards {
private:
    static const int totalCards = 52;
    std::vector<Card> deck;
    int currentCard;

public:
    DeckOfCards() {
        currentCard = 0;
        for (int count = 0; count < totalCards; ++count) {
            deck.push_back(Card(count % 13, count % 4));
        }
    }

    void shuffle() {
        std::default_random_engine rng(std::time(nullptr));
        std::shuffle(deck.begin(), deck.end(), rng);
```

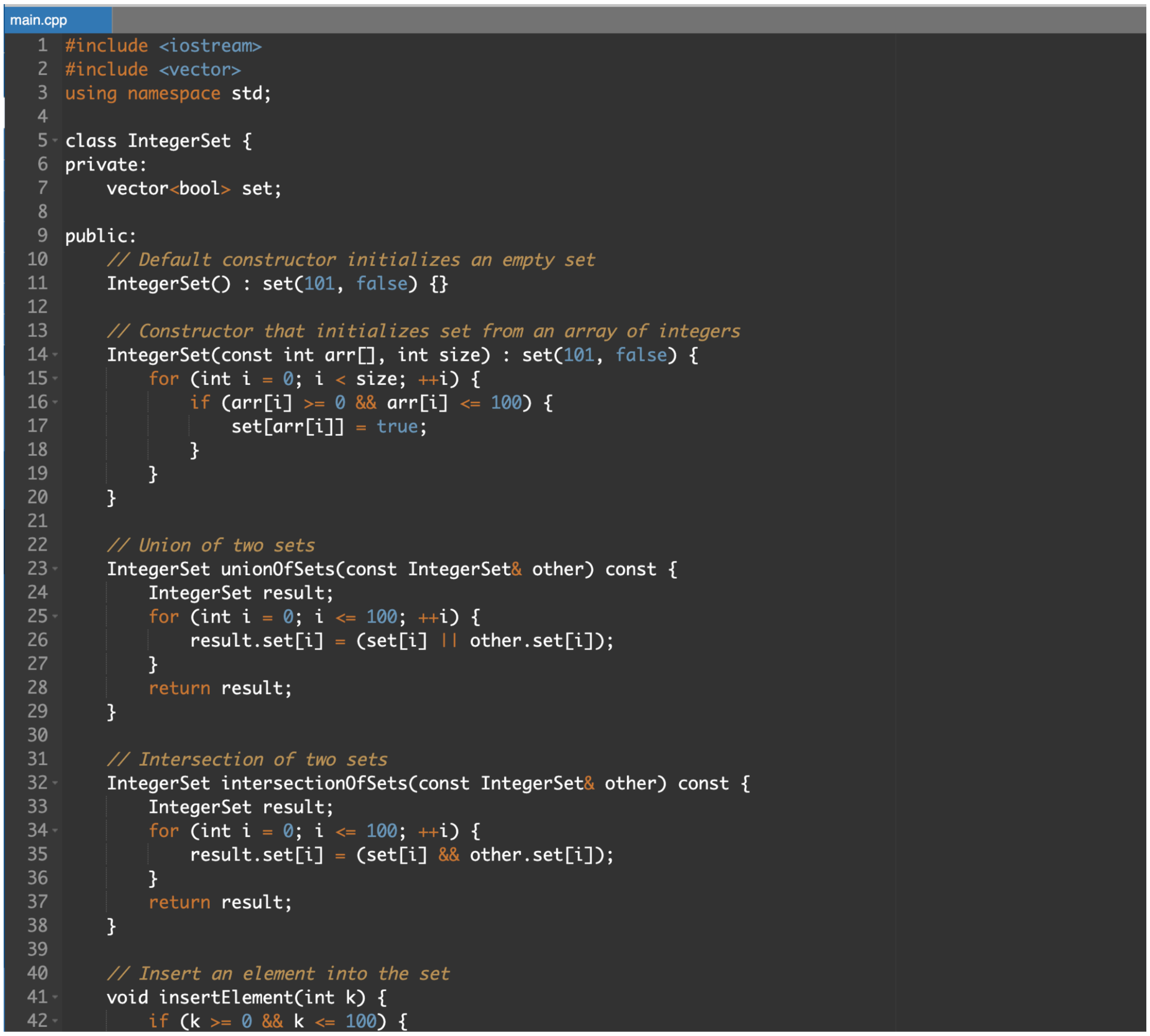


```
    }

    Card dealCard() {
        return deck[currentCard++];
    }

    bool moreCards() const {
        return currentCard < totalCards;
    }
};

int main() {
    DeckOfCards deck;
    deck.shuffle();

    std::cout << "Dealing the shuffled deck of cards:\n";
    while (deck.moreCards()) {
        std::cout << deck.dealCard().toString() << std::endl;
    }

    return 0;
}

```



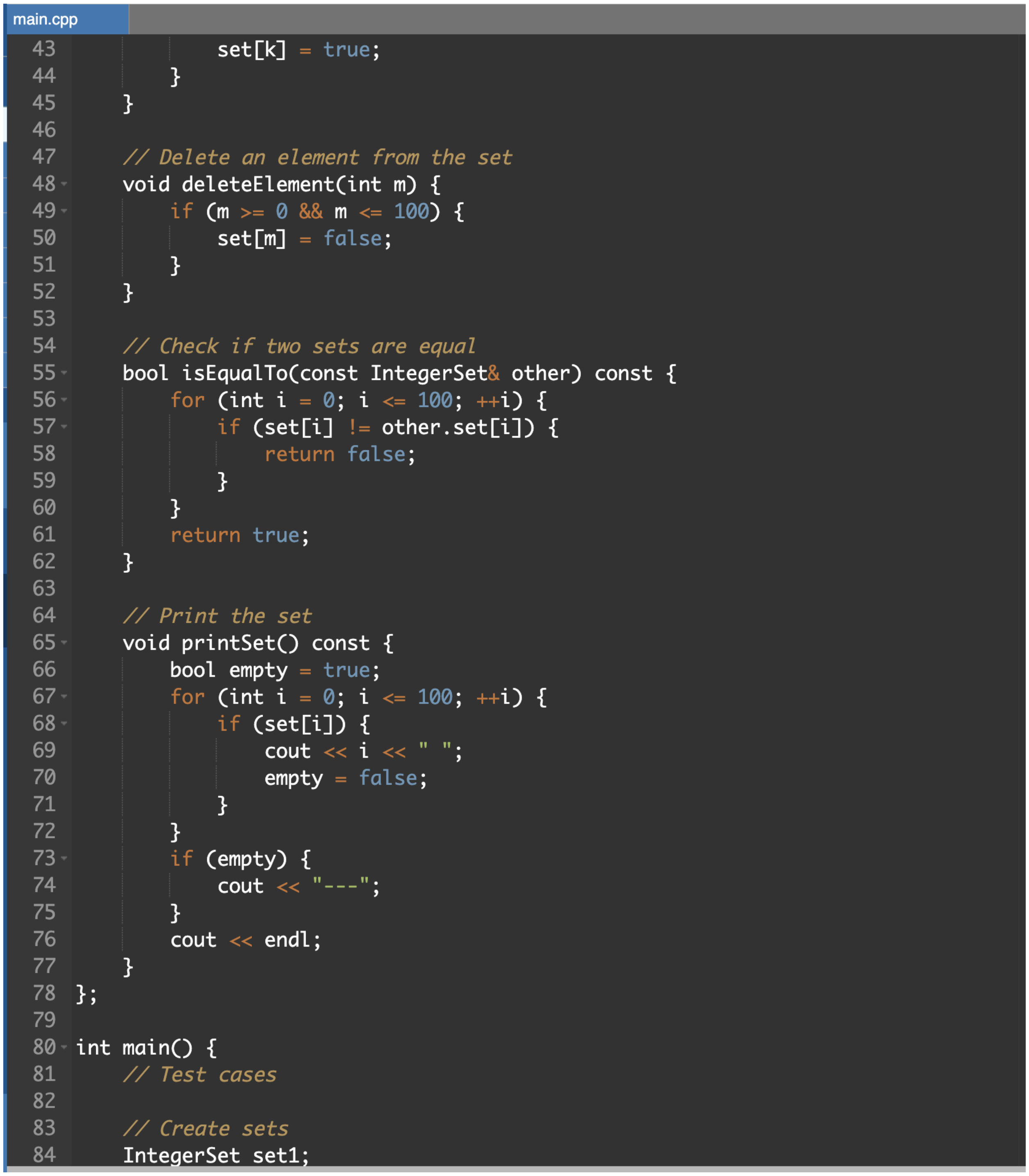
```
Dealing the shuffled deck of cards:
6 of Diamonds
Jack of Diamonds
3 of Clubs
3 of Hearts
8 of Hearts
5 of Spades
10 of Diamonds
4 of Diamonds
2 of Hearts
King of Clubs
5 of Hearts
10 of Clubs
Jack of Hearts
Jack of Clubs
4 of Clubs
9 of Hearts
Ace of Clubs
3 of Diamonds
7 of Clubs
Ace of Spades
Queen of Hearts
2 of Spades
Jack of Spades
5 of Clubs
King of Spades
Ace of Hearts
7 of Hearts
3 of Spades
10 of Spades
2 of Clubs
Queen of Diamonds
8 of Diamonds
6 of Hearts
7 of Diamonds
5 of Diamonds
6 of Spades
2 of Diamonds
9 of Spades
9 of Diamonds
10 of Hearts
8 of Clubs
Ace of Diamonds
8 of Spades
4 of Spades
7 of Spades
King of Diamonds
4 of Hearts
King of Hearts
9 of Clubs
Queen of Clubs
Queen of Spades
6 of Clubs
```

Question 2:

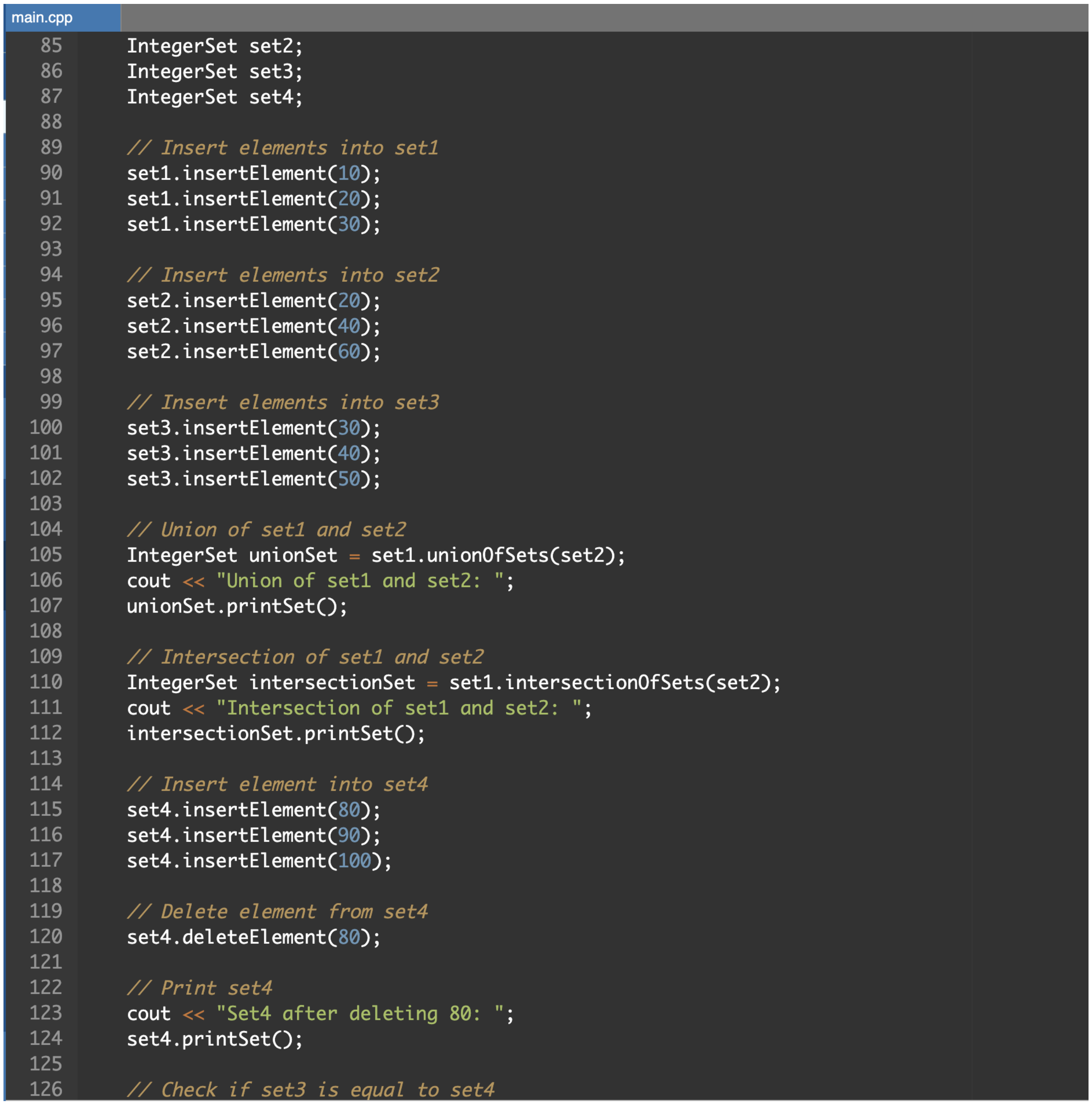


main.cpp

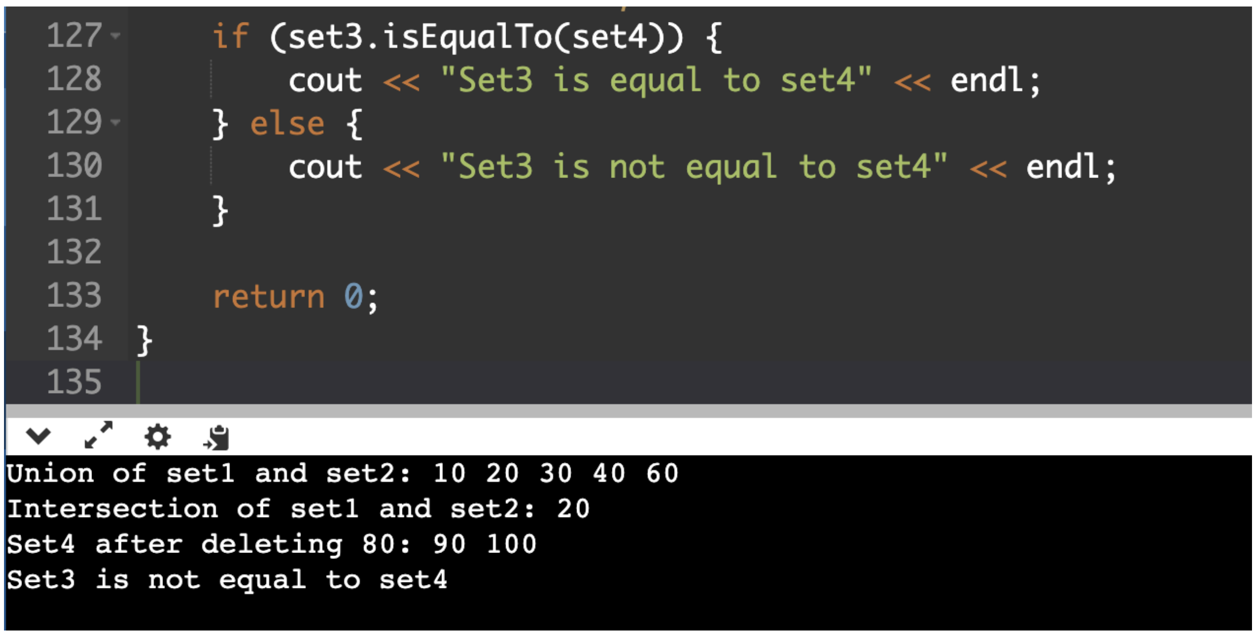
```cpp
#include <iostream>
#include <vector>
using namespace std;

class IntegerSet {
private:
    vector<bool> set;

public:
    // Default constructor initializes an empty set
    IntegerSet() : set(101, false) {}

    // Constructor that initializes set from an array of integers
    IntegerSet(const int arr[], int size) : set(101, false) {
        for (int i = 0; i < size; ++i) {
            if (arr[i] >= 0 && arr[i] <= 100) {
                set[arr[i]] = true;
            }
        }
    }

    // Union of two sets
    IntegerSet unionOfSets(const IntegerSet& other) const {
        IntegerSet result;
        for (int i = 0; i <= 100; ++i) {
            result.set[i] = (set[i] || other.set[i]);
        }
        return result;
    }

    // Intersection of two sets
    IntegerSet intersectionOfSets(const IntegerSet& other) const {
        IntegerSet result;
        for (int i = 0; i <= 100; ++i) {
            result.set[i] = (set[i] && other.set[i]);
        }
        return result;
    }

    // Insert an element into the set
    void insertElement(int k) {
        if (k >= 0 && k <= 100) {
```


main.cpp

```cpp
            set[k] = true;
        }
    }

    // Delete an element from the set
    void deleteElement(int m) {
        if (m >= 0 && m <= 100) {
            set[m] = false;
        }
    }

    // Check if two sets are equal
    bool isEqualTo(const IntegerSet& other) const {
        for (int i = 0; i <= 100; ++i) {
            if (set[i] != other.set[i]) {
                return false;
            }
        }
        return true;
    }

    // Print the set
    void printSet() const {
        bool empty = true;
        for (int i = 0; i <= 100; ++i) {
            if (set[i]) {
                cout << i << " ";
                empty = false;
            }
        }
        if (empty) {
            cout << "---";
        }
        cout << endl;
    }
};

int main() {
    // Test cases

    // Create sets
    IntegerSet set1;
```


main.cpp

```cpp
    IntegerSet set2;
    IntegerSet set3;
    IntegerSet set4;

    // Insert elements into set1
    set1.insertElement(10);
    set1.insertElement(20);
    set1.insertElement(30);

    // Insert elements into set2
    set2.insertElement(20);
    set2.insertElement(40);
    set2.insertElement(60);

    // Insert elements into set3
    set3.insertElement(30);
    set3.insertElement(40);
    set3.insertElement(50);

    // Union of set1 and set2
    IntegerSet unionSet = set1.unionOfSets(set2);
    cout << "Union of set1 and set2: ";
    unionSet.printSet();

    // Intersection of set1 and set2
    IntegerSet intersectionSet = set1.intersectionOfSets(set2);
    cout << "Intersection of set1 and set2: ";
    intersectionSet.printSet();

    // Insert element into set4
    set4.insertElement(80);
    set4.insertElement(90);
    set4.insertElement(100);

    // Delete element from set4
    set4.deleteElement(80);

    // Print set4
    cout << "Set4 after deleting 80: ";
    set4.printSet();

    // Check if set3 is equal to set4
```



```cpp
    if (set3.isEqualTo(set4)) {
        cout << "Set3 is equal to set4" << endl;
    } else {
        cout << "Set3 is not equal to set4" << endl;
    }

    return 0;
}
```

```
Union of set1 and set2: 10 20 30 40 60
Intersection of set1 and set2: 20
Set4 after deleting 80: 90 100
Set3 is not equal to set4
```