Question no.1

Create a program to shuffle and deal with a deck of cards. The program should consist of a class Card, a class DeckOfCards, and a main program. Class Card should provide:

a. Data member's face and suit of type int.
b. A constructor that receives two ints representing the face and suit and uses them to initialize the data members.
c. Two static arrays of strings representing the faces and suits.
d. A toString function that returns the Card as a string in the form "face of suit."

You can use the + operator to concatenate strings.

Class DeckOfCards should contain:

a. An array of Cards named a deck to store the Cards.
b. An integer current card represents the next card to deal.
c. A default constructor that initializes the Cards in the deck.
d. A shuffle function that shuffles the Cards in the deck. The shuffle algorithm should iterate through the array of Cards. For each Card, randomly select another Card in the deck and swap the two Cards.
e. A dealCard function that returns the next Card object from the deck.
f. A moreCards function that returns a bool value indicating whether there are more Cards to deal with.

The main program should create a DeckOfCards object, shuffle the cards, then deal the 52 cards.

- ⇨ Answer:
- ⇨ The code:

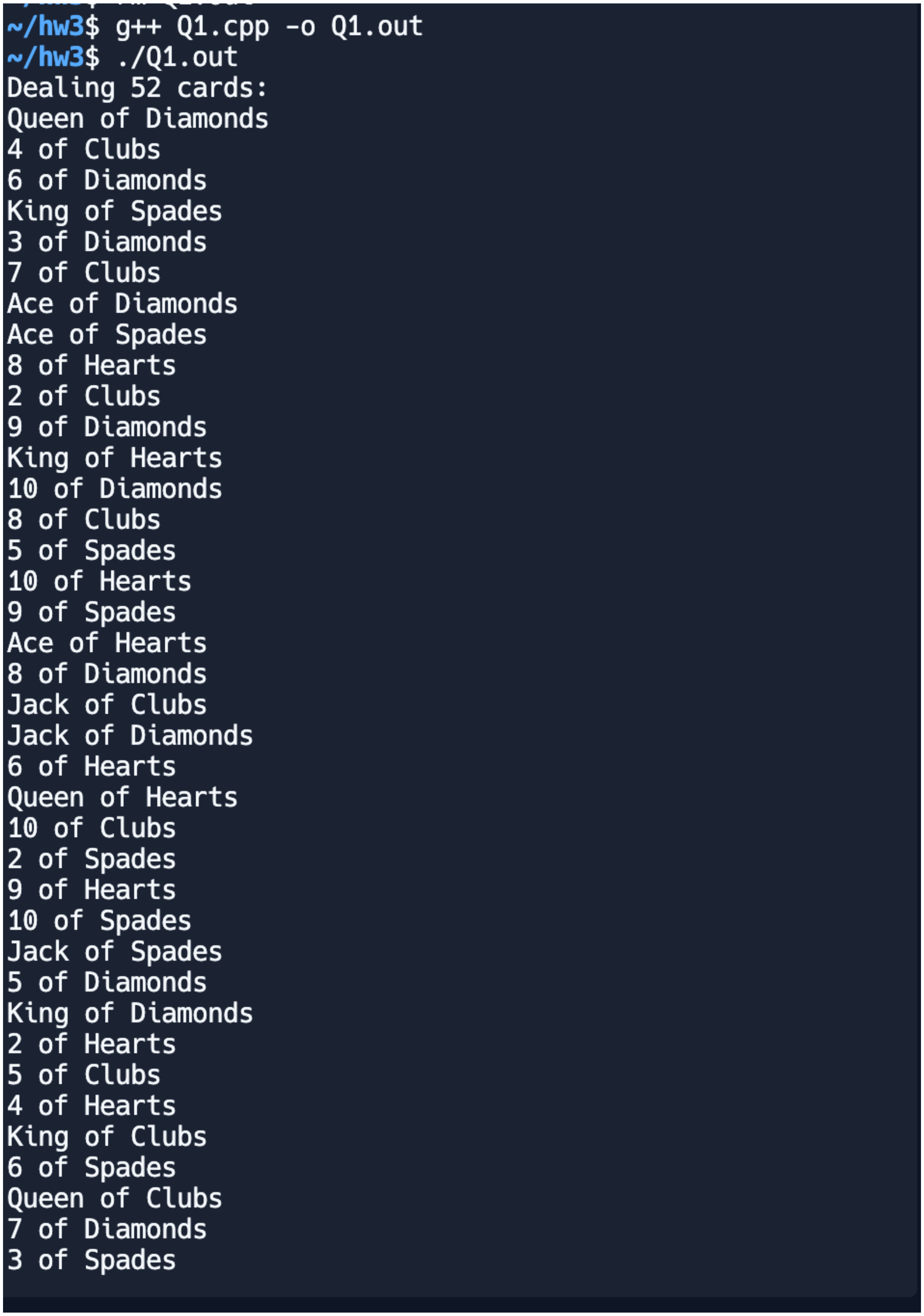
```
#include <iostream>
#include <string>
```

```
#include <cstdlib>
#include <ctime>

using namespace std;

class Card {
private:
   int face;
   int suit;
   static const string faces[13];
   static const string suits[4];
public:
   // Constructor with default arguments
   Card(int cardFace = 0, int cardSuit = 0) : face(cardFace),
suit(cardSuit) {}

   // Function to represent the card as a string
   string toString() const {
      return faces[face] + " of " + suits[suit];
   }
};

// Static arrays of strings representing faces and suits
const string Card::faces[13] = {"Ace", "2", "3", "4", "5", "6", "7", "8",
"9", "10", "Jack", "Queen", "King"};
const string Card::suits[4] = {"Hearts", "Diamonds", "Clubs",
"Spades"};

class DeckOfCards {
private:
   static const int totalCards = 52;
   Card deck[totalCards];
   int currentCard;
```

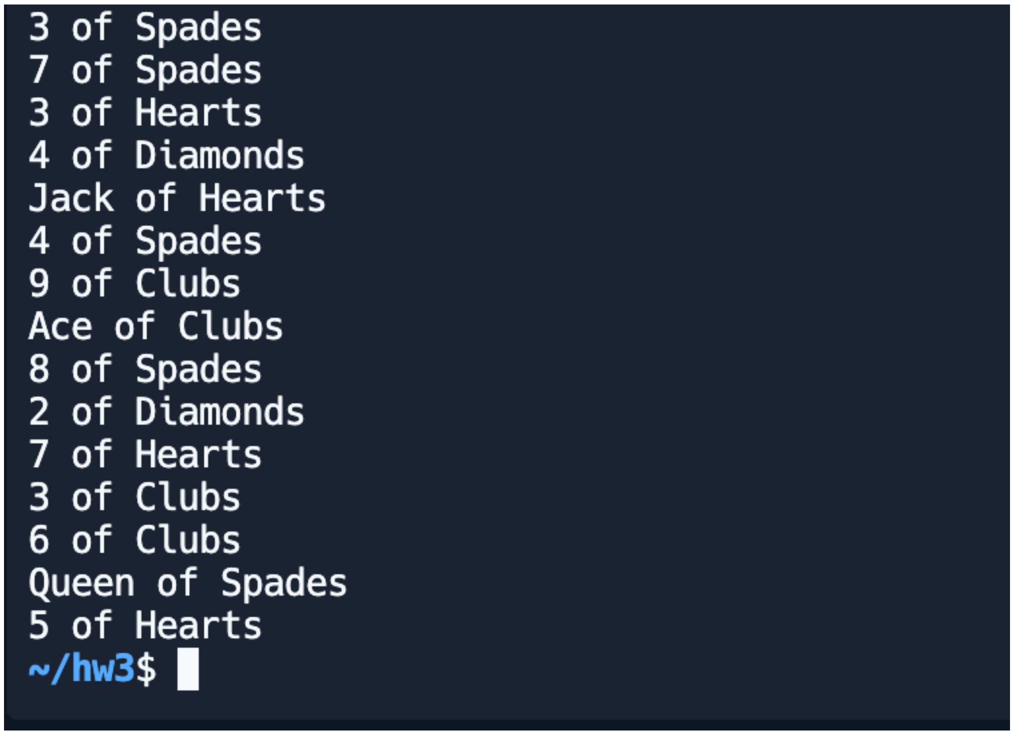
```cpp
public:
    // Default constructor to initialize the cards in the deck
    DeckOfCards() : currentCard(0) {
        for (int i = 0; i < totalCards; ++i) {
            deck[i] = Card(i % 13, i / 13);
        }
    }

    // Function to shuffle the cards in the deck
    void shuffle() {
        srand(time(0));
        for (int i = 0; i < totalCards; ++i) {
            int j = rand() % totalCards;
            swap(deck[i], deck[j]);
        }
        currentCard = 0;
    }

    // Function to deal a card from the deck
    Card dealCard() {
        if (moreCards()) {
            return deck[currentCard++];
        } else {
            cerr << "No more cards in the deck." << endl;
            return Card(-1, -1); // indicating an invalid card
        }
    }

    // Function to check if there are more cards to deal
    bool moreCards() const {
        return currentCard < totalCards;
    }
};
```

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

   cout << "Dealing 52 cards:\n";
   for (int i = 0; i < 52; ++i) {
      cout << deck.dealCard().toString() << endl;
   }

   return 0;
}
```

The output:



```
~/hw3$ g++ Q1.cpp -o Q1.out
~/hw3$ ./Q1.out
Dealing 52 cards:
Queen of Diamonds
4 of Clubs
6 of Diamonds
King of Spades
3 of Diamonds
7 of Clubs
Ace of Diamonds
Ace of Spades
8 of Hearts
2 of Clubs
9 of Diamonds
King of Hearts
10 of Diamonds
8 of Clubs
5 of Spades
10 of Hearts
9 of Spades
Ace of Hearts
8 of Diamonds
Jack of Clubs
Jack of Diamonds
6 of Hearts
Queen of Hearts
10 of Clubs
2 of Spades
9 of Hearts
10 of Spades
Jack of Spades
5 of Diamonds
King of Diamonds
2 of Hearts
5 of Clubs
4 of Hearts
King of Clubs
6 of Spades
Queen of Clubs
7 of Diamonds
3 of Spades
```



```
3 of Spades
7 of Spades
3 of Hearts
4 of Diamonds
Jack of Hearts
4 of Spades
9 of Clubs
Ace of Clubs
8 of Spades
2 of Diamonds
7 of Hearts
3 of Clubs
6 of Clubs
Queen of Spades
5 of Hearts
~/hw3$
```

Question no.2
Create class IntegerSet for which each object can hold integers in the range 0 through 100. Represent the set internally as a vector of bool values. Element a[i] is true if integer i is in the set. Element a[j] is false if integer j is not in the set. The default constructor initializes a set to the so-called "empty set," i.e., a set for which all elements contain false.

a. Provide member functions for the common set operations. For example, provide a unionOfSets member function that creates a third set that is the set- theoretic union of two existing sets (i.e., an element of the result is set to true if that element is true in either or both of the existing sets, and an element of the result is set to false if that element is false in each of the existing sets).

b. Provide an intersectionOfSets member function which creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the result is set to false if that element is false in either or both existing sets, and an element of the result is set to true if that element is true in each of the existing sets).

c. Provide an insertElement member function that places a new integer k into a set by setting a[k] to true. Provide a deleteElement member function that deletes integer m by setting a[m] to false.

d. Provide a printSet member function that prints a set as a list of numbers separated by spaces. Print only those elements that are present in the set (i.e., their position in the vector has a value of true). Print --- for an empty set.

e. Provide an isEqualTo member function that determines whether two sets are equal.

f. Provide an additional constructor that receives an array of integers and the size of that array and uses the array to initialize a set object.

Now write a main program to test your IntegerSet class. Instantiate several IntegerSet objects. Test that all your member functions work properly

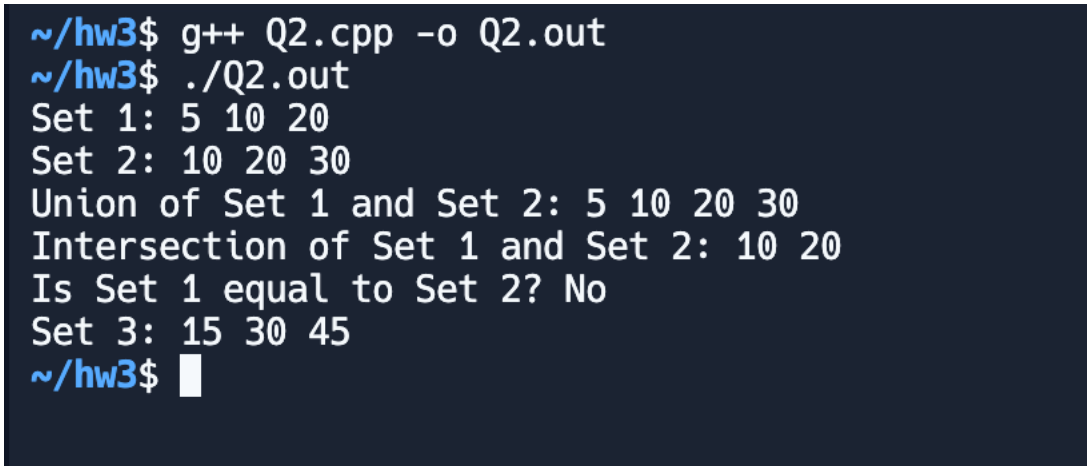
- ⇨ Answer:
- ⇨ The code:

```
#include <iostream>
#include <vector>

using namespace std;

class IntegerSet {
private:
    vector<bool> set; // Internal representation of the set as a
vector of bool values
public:
    // Default constructor initializes an empty set
    IntegerSet() : set(101, false) {}

    // Constructor to initialize set from an array of integers
    IntegerSet(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& otherSet) const {
        IntegerSet resultSet;
        for (int i = 0; i <= 100; ++i) {
            resultSet.set[i] = (set[i] || otherSet.set[i]); // Element is true
if present in either set
        }
        return resultSet;
    }

    // Intersection of two sets
    IntegerSet intersectionOfSets(const IntegerSet& otherSet)
const {
        IntegerSet resultSet;
        for (int i = 0; i <= 100; ++i) {
            resultSet.set[i] = (set[i] && otherSet.set[i]); // Element is
true if present in both sets
        }
    return resultSet;
}

// Insert element into the set
void insertElement(int k) {
    if (k >= 0 && k <= 100) {
        set[k] = true;
    }
}

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

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

// Check if two sets are equal
```

```
  bool isEqualTo(const IntegerSet& otherSet) const {
    for (int i = 0; i <= 100; ++i) {
      if (set[i] != otherSet.set[i]) {
        return false; // If any element is different, sets are not
equal
      }
    }
    return true; // All elements are same, sets are equal
  }
};

int main() {
  // Test IntegerSet class
  IntegerSet set1;
  set1.insertElement(5);
  set1.insertElement(10);
  set1.insertElement(20);

  IntegerSet set2;
  set2.insertElement(10);
  set2.insertElement(20);
  set2.insertElement(30);

  cout << "Set 1: ";
  set1.printSet();
  cout << "Set 2: ";
  set2.printSet();

  IntegerSet unionSet = set1.unionOfSets(set2);
  cout << "Union of Set 1 and Set 2: ";
  unionSet.printSet();

  IntegerSet intersectionSet = set1.intersectionOfSets(set2);
```

```
    cout << "Intersection of Set 1 and Set 2: ";
    intersectionSet.printSet();

    cout << "Is Set 1 equal to Set 2? " << (set1.isEqualTo(set2) ?
"Yes" : "No") << endl;

    // Test constructor with an array
    int arr[] = {15, 30, 45};
    IntegerSet set3(arr, 3);
    cout << "Set 3: ";
    set3.printSet();

    return 0;
}
```

⇨ The result:



```
~/hw3$ g++ Q2.cpp -o Q2.out
~/hw3$ ./Q2.out
Set 1: 5 10 20
Set 2: 10 20 30
Union of Set 1 and Set 2: 5 10 20 30
Intersection of Set 1 and Set 2: 10 20
Is Set 1 equal to Set 2? No
Set 3: 15 30 45
~/hw3$ 
```