San Francisco Bay University

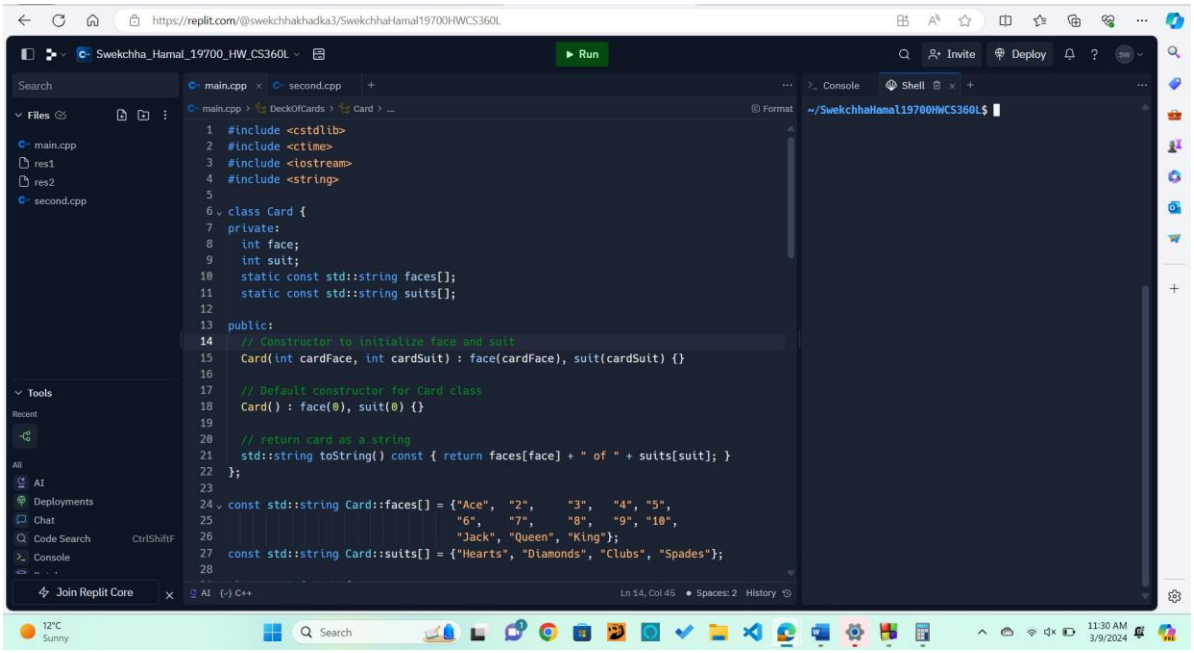
CS360 - Programming in C and C++
Homework Assignment #3

Due day: 3/9/2024

Instruction:

1. **Push the answer sheets/source code to Github**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue homework assignment submission can't be accepted.**
4. **Take academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**

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



   a. Data members *face* and *suit* of type *int*.

```
class Card {
private:
  int face;
  int suit;
  static const std::string faces[];
  static const std::string suits[];
```

b. A constructor that receives two *ints* representing the face and suit and uses them to initialize the data members.

```
public:
  // Constructor to initialize face and suit
  Card(int cardFace, int cardSuit) : face(cardFace), suit(cardSuit) {}
```

c. Two *static arrays* of *strings* representing the faces and suits.

```
static const std::string faces[];
static const std::string 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.

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

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

Class *DeckOfCards* should contain:

a. An *array* of *Cards* named *deck* to store the *Cards*.
b. An integer *currentCard* representing 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.

→

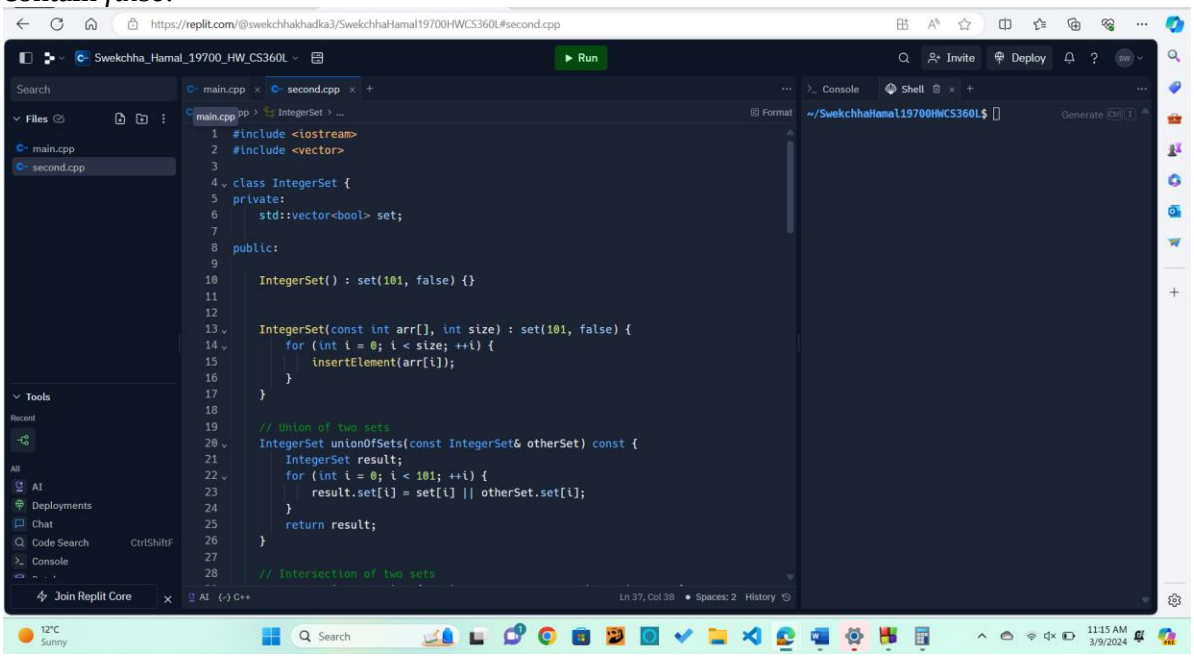
```
29  class DeckOfCards {
30  private:
31    Card deck[52];
32    int currentCard;
33
34  public:
35    // Default constructor to initialize the Cards in the deck
36    DeckOfCards() {
37      currentCard = 0;
38      for (int i = 0; i < 52; ++i) {
39        deck[i] = Card(i % 13, i / 13);
40      }
41    }
42
43    // Shuffle deck
44    void shuffle() {
45      for (int i = 0; i < 52; ++i) {
46        int j = rand() % 52;
47        Card temp = deck[i];
48        deck[i] = deck[j];
49        deck[j] = temp;
50      }
51    }
52
53    // next card from deck
54    Card dealCard() { return deck[currentCard++]; }
55
```

```
  // next card from deck
  Card dealCard() { return deck[currentCard++]; }

  // more cards to deal
  bool moreCards() const { return currentCard < 52; }
};

int main() {
  srand(static_cast<unsigned int>(time(0)));

  DeckOfCards deck;
  deck.shuffle();

  // Deal  52 cards
  while (deck.moreCards()) {
    std::cout << deck.dealCard().toString() << std::endl;
  }

  return 0;
}
```

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

```
~/SwekchhaHamal19700HWCS360L$ ./res1
5 of Hearts
5 of Diamonds
Jack of Spades
King of Diamonds
Ace of Spades
King of Clubs
4 of Diamonds
7 of Spades
9 of Spades
3 of Diamonds
5 of Spades
8 of Diamonds
7 of Hearts
Queen of Clubs
10 of Hearts
3 of Clubs
8 of Spades
7 of Diamonds
2 of Diamonds
5 of Clubs
Jack of Diamonds
Queen of Spades
Queen of Hearts
Queen of Diamonds
```

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).

```
// Union of two sets
IntegerSet unionOfSets(const IntegerSet& otherSet) const {
    IntegerSet result;
    for (int i = 0; i < 101; ++i) {
        result.set[i] = set[i] || otherSet.set[i];
    }
    return result;
}
```

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 of the existing sets, and an element of the result is set to *true* if that element is *true* in each of the existing sets).

```
// Intersection of two sets
IntegerSet intersectionOfSets(const IntegerSet& otherSet) const {
    IntegerSet result;
    for (int i = 0; i < 101; ++i) {
        result.set[i] = set[i] && otherSet.set[i];
    }
    return result;
}
```

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*.

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

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

```
// Print the set
void printSet() const {
    bool isEmpty = true;
    for (int i = 0; i < 101; ++i) {
        if (set[i]) {
            std::cout << i << " ";
            isEmpty = false;
        }
    }
    if (isEmpty) {
        std::cout << "---";
    }
    std::cout << std::endl;
}
```

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

```
    // Check if two sets are equal
    bool isEqualTo(const IntegerSet& otherSet) const {
        for (int i = 0; i < 101; ++i) {
            if (set[i] != otherSet.set[i]) {
                return false;
            }
        }
        return true;
    }
};
```

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.

```
IntegerSet(const int arr[], int size) : set(101, false) {
    for (int i = 0; i < size; ++i) {
        insertElement(arr[i]);
    }
}
```

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

```
int main() {
    IntegerSet set1, set2;
    set1.insertElement(10);
    set1.insertElement(25);
    set1.insertElement(35);

    set2.insertElement(10);
    set2.insertElement(45);
    set2.insertElement(50);

    IntegerSet unionSet = set1.unionOfSets(set2);
    IntegerSet intersectionSet = set1.intersectionOfSets(set2);

    std::cout << "Set 1: ";
    set1.printSet();
    std::cout << "Set 2: ";
    set2.printSet();
    std::cout << "Union of sets: ";
    unionSet.printSet();
    std::cout << "Intersection of sets: ";
    intersectionSet.printSet();

    std::cout << "Is set1 = set2? " << (set1.isEqualTo(set2) ? "Yes" : "No") <<
std::endl;

    return 0;
}
```

Output:

```
~/SwekchhaHamal19700HWCS360L$ g++ second.cpp -o res2
~/SwekchhaHamal19700HWCS360L$ ./res2
Set 1: 10 25 35
Set 2: 10 45 50
Union of sets: 10 25 35 45 50
Intersection of sets: 10
Is set1 = set2? No
```