TASMITA TANJIM (19723)

REPLIT LINK:
https://replit.com/@TASMITA-TANJIMT/HW3-1

QUESTION NO 01:

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.
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.

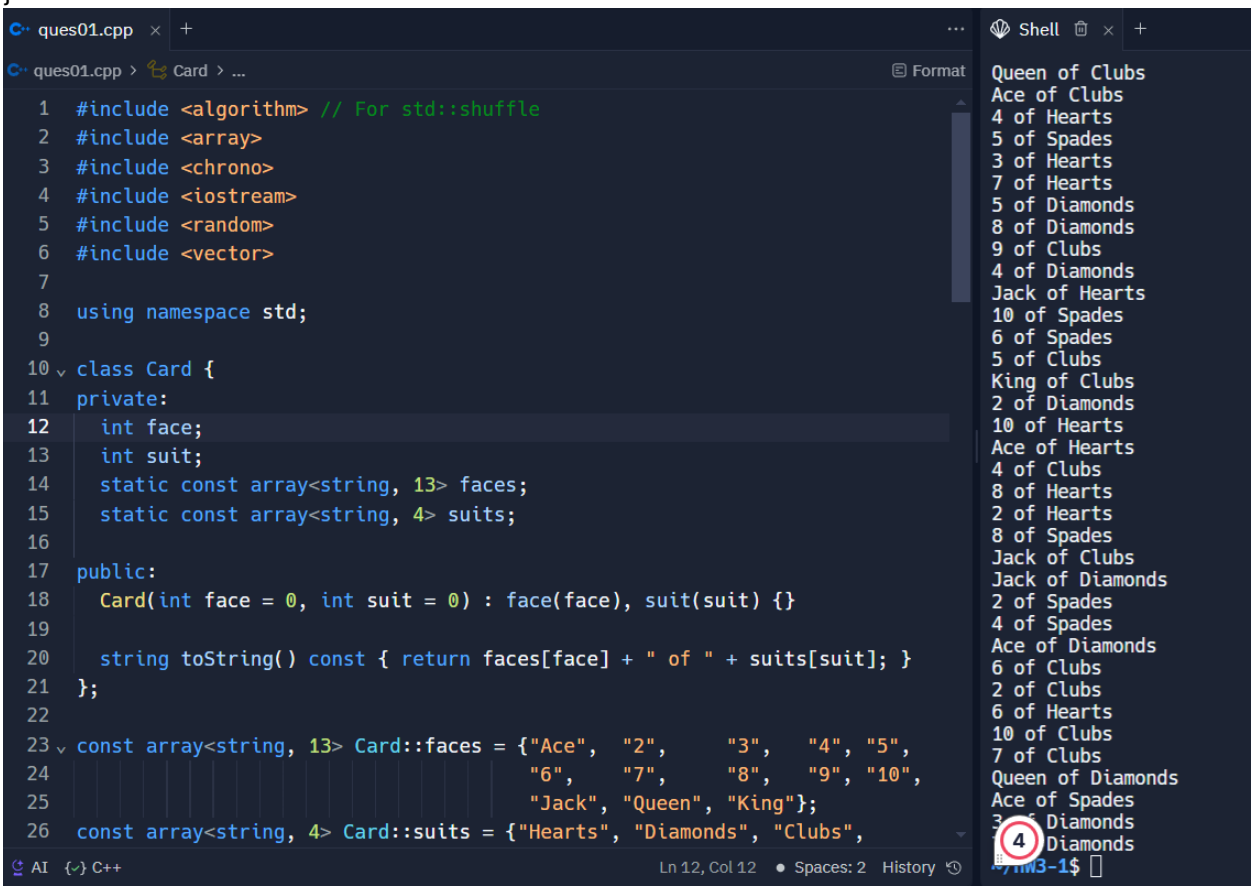
Class DeckOfCards should contain:
a. An array of Cards named deck to store the Cards.
b. An integer currentCard representing the next card to deal.
c. A default constructor that initializes the Cards in the deck.
d. A shuffle function that shuffles the Cards in the deck. The shuffle algorithm should 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.
The main program should create a DeckOfCards object, shuffle the cards, then deal the 52 cards.

CODE:

```
#include <algorithm> // For std::shuffle
#include <array>
#include <chrono>
#include <iostream>
#include <random>
#include <vector>

using namespace std;

class Card {
private:
 int face;
 int suit;
 static const array<string, 13> faces;
 static const array<string, 4> suits;

public:
 Card(int face = 0, int suit = 0) : face(face), suit(suit) {}

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

```
};

const array<string, 13> Card::faces = {"Ace",  "2",    "3",   "4", "5",
                                       "6",    "7",    "8",   "9", "10",
                                       "Jack", "Queen", "King"};
const array<string, 4> Card::suits = {"Hearts", "Diamonds", "Clubs", "Spades"};

class DeckOfCards {
private:
 vector<Card> deck;
 int currentCard;

public:
 DeckOfCards() : currentCard(0) {
  deck.reserve(52);
  for (int suit = 0; suit < 4; ++suit) {
   for (int face = 0; face < 13; ++face) {
    deck.emplace_back(face, suit);
   }
  }
 }

 void shuffle() {
  unsigned seed = chrono::system_clock::now().time_since_epoch().count();
  std::shuffle(deck.begin(), deck.end(),
           std::default_random_engine(
              seed)); // std:: to specify the standard library
 }

 Card dealCard() {
  if (currentCard < deck.size()) {
   return deck[currentCard++];
  } else {
   return Card(-1, -1); // to indicate no more cards
  }
 }

 bool moreCards() const { return currentCard < deck.size(); }
};

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

 while (deckOfCards.moreCards()) {
  Card dealtCard = deckOfCards.dealCard();
  cout << dealtCard.toString() << endl;
 }
```

return 0;
}



ANSWER OF 02:

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 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).
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

CODE:

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

using namespace std;

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

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

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

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

```
    // 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;
        }
    }

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

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

private:
    vector<bool> set;
};

int main() {
    // Example usage
    IntegerSet set1;
    set1.insertElement(34);
    set1.insertElement(56);
```

```
    set1.insertElement(78);
    cout << "Set1 after insertions: ";
    set1.printSet();

    int elements[] = {34, 78, 90};
    IntegerSet set2(elements, 3);
    cout << "Set2 initialized with elements: ";
    set2.printSet();

    IntegerSet unionSet = set1.unionOfSets(set2);
    cout << "Union of Set1 and Set2: ";
    unionSet.printSet();

    IntegerSet intersectionSet = set1.intersectionOfSets(set2);
    cout << "Intersection of Set1 and Set2: ";
    intersectionSet.printSet();

    set1.deleteElement(56);
    cout << "Set1 after deletion of 56: ";
    set1.printSet();

    bool isEqual = set1.isEqualTo(set2);
    cout << "Set1 is equal to Set2: " << (isEqual ? "true" : "false") << endl;

    return 0;
}
```