CS 7B - Fall '19 - Project 1: The 21 Card Trick

This assignment is designed to provoke thought and engender understanding of how classes and objects can be developed to model a deck of playing cards and then to use this simulate playing the so-called "21 Card Trick." This assignment must be completed individually.

Due Wednesday, September 18 by the start of lecture.
Working in groups is not permitted

We'll take as a point of departure a question raised in the "codereview" section of the stackexchange.com website: /https://codereview.stackexchange.com/questions/41810/card-and-deck-classes.

1. The questioner (one "CroCo") proposes working code for files `card.h` and `deck.h` and asks,

   > "I would like to know whether I'm going in a correct way or not in building my Poker game. So far, I've implemented card and deck classes and I would like to see your feedback about my work. Feel free to criticize the code of any regard (organization, order, comments ... etc)"

   Below are listings of the code he posts.

   First we have the `Card` class interface, which declares global constants for `SUIT_MAX` and `RANK_MAX`, lends the `Deck` class `friend` status, declares a default constructor and a constructor taking references to `const int` objects and a `card2Str()` function. Then there are a slew of 4 private functions for handling suits and ranks and `int` variables for `m_suit` and `m_rank`:

```
1  // card.h
   //--------------------------
3  #pragma once
   #include <string>
5
   const int SUIT_MAX(4);
7  const int RANK_MAX(13);

9  class Card
   {
11     friend class Deck; // Deck Class needs to access to Card Class but not vice versa
   public:
13     explicit Card();
       explicit Card(const int &suit, const int &rank);
15
       std::string Card2Str() const;
17
   private:
19     int generate_suit();
       int generate_rank();
21     int get_suit() const;
       int get_rank() const;
23     int m_suit;
       int m_rank;
25 };
```

   This is followed, appropriately by the `Card` class implementation, defining a few global `string` arrays the various functions declared in the interface:

```cpp
// card.cpp
//----------------------------------------
#include <stdlib.h>      /* srand, rand */
#include "card.h"
#include <iostream>

const std::string SUIT[SUIT_MAX]  = {"S", "H", "D", "C"};
const std::string RANK[RANK_MAX]  = {"2","3","4","5","6","7","8","9","10","J","Q","K",
    "A"};

Card::Card() {
   m_suit = generate_suit();
   m_rank = generate_rank();
}

Card::Card(const int &suit, const int &rank) : m_suit(suit), m_rank(rank) {}

int Card::generate_suit() {
    return rand() % (SUIT_MAX-1) + 0;
}

int Card::generate_rank() {
    return rand() % (RANK_MAX-1) + 0;
}

std::string Card::Card2Str() const {
    return SUIT[get_suit()] + RANK[get_rank()];
}

int Card::get_suit() const {
    return m_suit;
}

int Card::get_rank() const {
    return m_rank;
}
```

Then we have the `Deck` class interface, representing the deck of cards as a `vector<Card>` together with prototypes for methods of constructing and printing cards:

```cpp
// deck.h
//----------------------------------------
#pragma once
#include <vector>
#include <iostream>
#include <fstream>
#include "card.h"

class Deck {
public:
      explicit Deck();
      void print_Deck() const;
      void getOneCard();
private:
    std::vector<Card> m_deck;

};
```

This is followed by the implementation of the `Deck` class functions:

```
#include <iostream>
#include "deck.h"

Deck::Deck() {
    for (unsigned int i(0); i < SUIT_MAX; ++i)      {
        for (unsigned int j(0); j < RANK_MAX; ++j)          {
            Card card(i, j);
            m_deck.push_back(card);
        }
    }
}


void Deck::print_Deck() const {
    unsigned int count(1);

    for (unsigned int i(0); i < m_deck.size(); ++i)      {
        std::cout << m_deck[i].Card2Str() << " ";
        if ( count == 13 )          {
          std::cout << std::endl;
          count = 0;
        }
        ++count;
    }
}

void Deck::getOneCard() {
    Card cd(m_deck.back().get_suit(), m_deck.back().get_rank());
    m_deck.pop_back();
    std::cout << cd.Card2Str() << std::endl;
}
```

Finally, the driver to test these classes work well:

```
#include <iostream>
#include <vector>
#include <stdlib.h>      /* srand, rand */
#include <time.h>        /* time */
#include <string>

#include "card.h"
#include "deck.h"

int main()  {
    srand (time(NULL));

    Deck _deck;
    _deck.print_Deck();
    _deck.getOneCard();

    std::cout << std::endl;
    _deck.print_Deck();
}
```

This code leaves much room for improvement, but it does compile and run, producing this output:

```
S2 S3 S4 S5 S6 S7 S8 S9 S10 SJ SQ SK SA
H2 H3 H4 H5 H6 H7 H8 H9 H10 HJ HQ HK HA
D2 D3 D4 D5 D6 D7 D8 D9 D10 DJ DQ DK DA
C2 C3 C4 C5 C6 C7 C8 C9 C10 CJ CQ CK CA
CA

S2 S3 S4 S5 S6 S7 S8 S9 S10 SJ SQ SK SA
H2 H3 H4 H5 H6 H7 H8 H9 H10 HJ HQ HK HA
D2 D3 D4 D5 D6 D7 D8 D9 D10 DJ DQ DK DA
C2 C3 C4 C5 C6 C7 C8 C9 C10 CJ CQ CK
```

The cards are printed (whole deck from Deuce of Spades, S2, to Ace of Clubs, CA), then the Ace of Clubs is dealt from the top of the deck and the whole remaining deck printed out again.

Now in answer to CroCo's question, Stackexchange moderator Jamal has some constructive feedback, starting with the recommendation that the random number generator can, since C++11, be seeded with `srand(time(nullptr))`–we'll come back to the random number generator later.

Also, `stdlib.h` and `time.h` are both C libraries. The respective C++ libraries `cstdlib` and `ctime` are preferred here, and you don't need to include `vector` in `main.cpp`, since it's already included in `deck.h`.

The moderator also recommends using an iterator to traverse the `vector<Card>` used to contain a `Deck`, either be dereferencing it like so:

```
for (auto iter = m_deck.begin(); iter != m_deck.cend(); ++iter) {
    std::cout << *iter << "\n";
}
```

...or, with C++11, we can use a range-based for-loop:

Here we're using `auto` to detect the type of the variable `iter`, based on the assignment of `m_deck.begin()`, which is of type pointer to vector element. Note that we dereference `iter` with `*iter` to get the value of the thing it points to, which, since `m_deck` is a `vector<Card>` is a `Card` object. This presents a problem, since nowhere in the code do we provide a method for inserting a `Card` into the `cout` stream.

Following the discussion here, we can use a friend function like so (declared in the `card.h` file and defined in the `card.cpp` file):

```
friend std::ostream& Card::operator<<(std::ostream& stream, Card const& card) {
    return stream << std::setw(17) << card.getRankStr() + "of" + card.getSuitStr();
}
```

...or, alternatively, a helper function defined outside the class:

```
std::ostream& operator<<(std::ostream& stream, Card const& card)  {
    return stream << std::setw(17) << card.getRankStr() +"of" + card.getSuitStr();
}
```

...but, we haven't defined `getRankStr()` nor `getSuitStr()`, yet, so we're getting a bit ahead of ourselves.

Understandably, moderator Jamal takes great exception to CroCo's Card constructor, writing,

> "I don't know why you're using randomness to generate a card within the class. The constructor should be given input to determine its rank and suit. A card shouldn't create itself, and you wouldn't want this with an actual card game. Remove the generate functions entirely, and instead allow the constructor to receive arguments to determine the card's value.
>
> "You also don't need to construct the deck within its constructor. A deck can still start out empty until cards are passed to it. That should be handled in client code, especially if someone doesn't want a 52-card deck (or a certain game uses something different).
>
> "As an alternative to using `std::strings`, consider `enum`s for the ranks and suits:"

```
enum Rank { Ace=0, Deuce, Trey, Four, /* ... */, Queen, King };
enum Suit { Clubs, Hearts, Spades, Diamonds };
```

Jamal advises,

> "With this, you'll no longer need the arrays nor the max constants. Your `m_rank` and `m_suit` members should also be one of these `enum` type (`Rank` and `Suit` respectively). The `enum`s and the `Card` class declaration should all then be contained in a namespace.
>
> Within your `Card` class, you'll need functions to convert a given `enum` value to a `char` or `string` value for displaying. You could, for instance, do this with a switch statement, which will serve as a look-up table. This will also make it easier to maintain the ranks and suits.

```
std::string getRank(Rank rank) {
    switch (rank) {
    case Ace:
        return "Ace";
    case Deuce:
        return "Deuce";
        // ...
    default: throw std::logic_error("invalid rank");
    }
}
// same idea with suits
```

Now, as to the `Card2Str()` function, I recommend declaring the member functions below in `card.h`:

```
std::string getRankStr() const;
std::string getSuitStr() const;
Rank getRank() const;
Suit getSuit() const;
```

...and define them in `card.cpp`;

When you put these improvements all together, this driver code:

```
#include "Card.h"
#include "Deck.h"

int main()
{
  Deck deck;
  deck.printDeck();
  deck.getOneCard();

  std::cout << std::endl;
  deck.printDeck();

  std::cout << "\nThe top card is " << deck.top();
  std::cout << std::endl;
  deck.printDeck();

  return 0;
}
```

will produce an output like this:

```
AceofClubs        DeuceofClubs      TreyofClubs       FourofClubs
FiveofClubs       SixofClubs        SevenofClubs      EightofClubs
NineofClubs       TenofClubs        JackofClubs       QueenofClubs
KingofClubs       AceofHearts       DeuceofHearts     TreyofHearts
FourofHearts      FiveofHearts      SixofHearts       SevenofHearts
EightofHearts     NineofHearts      TenofHearts       JackofHearts
QueenofHearts     KingofHearts      AceofSpades       DeuceofSpades
TreyofSpades      FourofSpades      FiveofSpades      SixofSpades
SevenofSpades     EightofSpades     NineofSpades      TenofSpades
JackofSpades      QueenofSpades     KingofSpades      AceofDiamonds
DeuceofDiamonds   TreyofDiamonds    FourofDiamonds    FiveofDiamonds
SixofDiamonds     SevenofDiamonds   EightofDiamonds   NineofDiamonds
TenofDiamonds     JackofDiamonds    QueenofDiamonds   KingofDiamonds
KingofDiamonds

AceofClubs        DeuceofClubs      TreyofClubs       FourofClubs
FiveofClubs       SixofClubs        SevenofClubs      EightofClubs
NineofClubs       TenofClubs        JackofClubs       QueenofClubs
KingofClubs       AceofHearts       DeuceofHearts     TreyofHearts
FourofHearts      FiveofHearts      SixofHearts       SevenofHearts
EightofHearts     NineofHearts      TenofHearts       JackofHearts
QueenofHearts     KingofHearts      AceofSpades       DeuceofSpades
TreyofSpades      FourofSpades      FiveofSpades      SixofSpades
SevenofSpades     EightofSpades     NineofSpades      TenofSpades
JackofSpades      QueenofSpades     KingofSpades      AceofDiamonds
DeuceofDiamonds   TreyofDiamonds    FourofDiamonds    FiveofDiamonds
SixofDiamonds     SevenofDiamonds   EightofDiamonds   NineofDiamonds
TenofDiamonds     JackofDiamonds    QueenofDiamonds
The top card is QueenofDiamonds
AceofClubs        DeuceofClubs      TreyofClubs       FourofClubs
```

```
FiveofClubs       SixofClubs        SevenofClubs      EightofClubs
NineofClubs       TenofClubs        JackofClubs       QueenofClubs
KingofClubs       AceofHearts       DeuceofHearts     TreyofHearts
FourofHearts      FiveofHearts      SixofHearts       SevenofHearts
EightofHearts     NineofHearts      TenofHearts       JackofHearts
QueenofHearts     KingofHearts      AceofSpades       DeuceofSpades
TreyofSpades      FourofSpades      FiveofSpades      SixofSpades
SevenofSpades     EightofSpades     NineofSpades      TenofSpades
JackofSpades      QueenofSpades     KingofSpades      AceofDiamonds
DeuceofDiamonds   TreyofDiamonds    FourofDiamonds    FiveofDiamonds
SixofDiamonds     SevenofDiamonds   EightofDiamonds   NineofDiamonds
TenofDiamonds     JackofDiamonds    QueenofDiamonds
```

Finally, write code to simulate the play of the 21 Card Trick. This is fairly wide-open, but use the `Card` and `Deck` classes you've created. Here's some pseudocode to get you started. Note that the more you "divide and conquer" these operations into simple function calls, the better.

```
#include "Card.h"
#include "Deck.h"

int main() {
  //seed the random number generator
  Deck deck;
  //limit your deck to some (random?) group of 21 cards
  while (1) {
    for (int i = 0; i < 3; ++i) {
      //Shuffle 21 cards and display (requires use of random numbers).
      //Ask subject to chose a card from the 21 but not to tell which it is.
      //Deal three groups of seven cards each, from left to right in threes
          //and then down seven times.
      //Ask subject to chose the group that includes their card
          //(number between 1 and 3, inclusive).
      //Collect cards so that the chosen group is in the middle.
      //Make a note of which cards are possibly the subject's card.
    }
    // after the third go there will be only one possibility, ask the subject
    // if this is their card and give an appropriate response to
    // their yes/no reply
  }
}
```