17

```
#include <iostream>
#include <string>
#include <ctime>
#include <list>

using namespace std;

class Card
{
public:
   enum Suit { clubs, diamonds, hearts, spades };
   static char DEFAULT_VAL;
   static Suit DEFAULT_SUIT;

private:
   char value;
   Suit suit;
   bool errorFlag;

   bool isValid(char value, Suit suit);

public:
   static const int NUM_CARD_VALS = 13;
   static const int NUM_CARD_SUITS = 4;
   const static char valueRanks[NUM_CARD_VALS];
   const static Suit suitRanks[NUM_CARD_SUITS];

   int compareTo(const Card& other) const;

   static int getSuitRank(Suit st);
   static int getValueRank(char val);

   Card(char value = DEFAULT_VAL, Suit suit = DEFAULT_SUIT);
   string toString() const;
   bool set(char values = DEFAULT_VAL, Suit suit = DEFAULT_SUIT);

   char getVal() { return value; }
   Suit getSuit() { return suit; }
   bool getErrorFlag() { return errorFlag; }
   bool equals(Card card);
};

typedef list<Card> CardList;
void showList(const CardList& myList);
void insert(CardList& myList, Card& x);
bool remove(CardList& myList, Card& x);
bool removeAll(CardList& myList, Card& x);
// for easy comparisons
int operator==(const Card& first, const Card& other) { return first.compareTo(ot
her) == 0; }
// for client Card generation
Card generateRandomCard();
char Card::DEFAULT_VAL = 'A';
Card::Suit Card::DEFAULT_SUIT = Card::spades;
// for comparisons -- ordering values and ranks
const char Card::valueRanks[NUM_CARD_VALS] // const forces correct # initializer
s
= { '2', '3', '4', '5', '6', '7', '8', '9', 'T',
'J', 'Q', 'K', 'A' };
const Card::Suit Card::suitRanks[NUM_CARD_SUITS] =
{
   Card::clubs, Card::diamonds,
```

const -2

```
   Card::hearts, Card::spades
};

int main()
{
   Card card1, card2;
   srand(time(NULL)); // or not, if you want repetition
   cout << "should all be 0:\n";
   card1.set('A', Card::spades); card2.set('A', Card::spades);
   cout << card1.compareTo(card2) << endl;
   card1.set('4', Card::hearts); card2.set('4', Card::hearts);
   cout << card1.compareTo(card2) << endl;
   card1.set('T', Card::clubs); card2.set('T', Card::clubs);
   cout << card1.compareTo(card2) << endl;
   cout << "should all be < 0 :\n";
   card1.set('A', Card::clubs); card2.set('A', Card::spades);
   cout << card1.compareTo(card2) << endl;;
   card1.set('4', Card::hearts); card2.set('5', Card::hearts);
   cout << card1.compareTo(card2) << endl;
   card1.set('9', Card::hearts); card2.set('T', Card::clubs);
   cout << card1.compareTo(card2) << endl;
   cout << "should all be > 0 :\n";
   card1.set('A', Card::clubs); card2.set('K', Card::clubs);
   cout << card1.compareTo(card2) << endl;
   card1.set('6', Card::hearts); card2.set('5', Card::spades);
   cout << card1.compareTo(card2) << endl;
   card1.set('K', Card::diamonds); card2.set('K', Card::clubs);
   cout << card1.compareTo(card2) << endl;
   cout << "\nSome random cards:\n";
   int counter = 0;
   for (int k = 0; k < 50; k++)
   {
      cout << generateRandomCard().toString() << " ";
      if (counter == 4)
      {
         counter = 0;
         cout << endl;
      }
      else
      {
         counter++;
      }
   }
   cout << endl << endl;

   cout << "-------------------- Card List --------------------\n\n";

   CardList CL;
   for (int i = 0; i < 10; i++)
   {
    Card randCard = generateRandomCard();
    insert(CL, randCard);
    insert(CL, randCard);
   }
   showList(CL);
   cout << endl;

   cout << "-------------------- Cards After Removed ---------------------\n\n";

   for (int j = 0; j < 5; j++)
   {
      Card card3 = *CL.begin();
```

use proper CamelCase names

-1

```
      while (remove(CL, card3))
      {
         remove(CL, card3);
      }
   }

   showList(CL);
   cout << endl;

   Card card3 = *CL.begin();

   removeAll(CL, card3);
   cout << "--------------------- All Of One Card Removed --------------------\n
\n";
   showList(CL);

   cout << endl << endl;
   return 0;
}

Card::Card(char value, Suit suit)
{
   set(value, suit);
}

string Card::toString() const
{
   string retVal = " ";

   if (errorFlag)
      return "** Illegal **";

   retVal[0] = value;

   if (suit == spades)
      retVal += " of Spades";
   else if (suit == hearts)
      retVal += " of Hearts";
   else if (suit == diamonds)
      retVal += " of Diamonds";
   else if (suit == clubs)
      retVal += " of Clubs";

   return retVal;
}

bool Card::set(char value, Suit suit)
{
   char upVal;

   upVal = toupper((int)value);

   if (!isValid(upVal, suit))
   {
      errorFlag = true;
      return false;
   }
   errorFlag = false;
   this->value = upVal;
   this->suit = suit;
   return true;
```

```
}

bool Card::isValid(char value, Suit suit)
{
   string upVal = "_";
   string legalVals = "23456789TJQKA";

   upVal[0] = toupper((int)value);

   return legalVals.find(upVal) != string::npos;
}

bool Card::equals(Card card)
{
   if (this->value != card.value)
      return false;
   if (this->suit != card.suit)
      return false;
   if (this->errorFlag != card.errorFlag)
      return false;
   return true;
}

int Card::compareTo(const Card& other) const {
   if (this->value == other.value)
      return getSuitRank(this->suit) - getSuitRank(other.suit);
   return getValueRank(this->value) - getValueRank(other.value);
}

int Card::getSuitRank(Suit st) {
   for (int k = 0; k < NUM_CARD_SUITS; k++)
      if (suitRanks[k] == st)
         return k;
   return 0; // should not happen
}

int Card::getValueRank(char val) {
   for (int k = 0; k < NUM_CARD_VALS; k++)
      if (valueRanks[k] == val)
         return k;
   return 0; // should not happen
}

Card generateRandomCard() {
   Card::Suit suit = (Card::Suit) (rand() % Card::NUM_CARD_SUITS);
   char val = Card::valueRanks[rand() % Card::NUM_CARD_VALS];
   return Card(val, suit);
}

void showList(const CardList& myList)
{
   list<Card>::iterator iter;
   int counter = 0;
   for (list<Card>::const_iterator iter = myList.begin(); iter != myList.end();
iter++)
   {
      cout << (*iter).toString() << " ";
      if (counter == 4)
      {
         counter = 0;
         cout << endl;
      }
```

use the modulus op.

```
      else
      {
         counter++;
      }
   }
}

void insert(CardList& myList, Card& x)
{
   list<Card>::iterator iter;

   // loop until we find a float > x
   for (iter = myList.begin(); iter != myList.end(); iter++)
      if (x.compareTo (*iter) < 0)
         break;   // found the exact place for this float
   myList.insert(iter, x);
}
bool remove(CardList& myList, Card& x)
{
   list<Card>::iterator iter;

   // loop until we find or exhaust list
   for (iter = myList.begin(); iter != myList.end(); iter++)
      if (x.equals (*iter))
      {
         myList.erase(iter);
         return true;
      }
   return false;
}
bool removeAll(CardList& myList, Card& x)
{
   list<Card>::iterator iter;

   for (iter = myList.begin(); iter != myList.end(); iter++)
      if (x.equals(*iter))
      {
         myList.remove(*iter);
         return true;
      }
   return false;
}

/*----------------------------- Posted Run ------------------------------------
-
should all be 0:
0
0
0
should all be < 0 :
-3
-1
-1
should all be > 0 :
1
1
1

Some random cards:
2 of Spades 8 of Hearts 3 of Diamonds 9 of Hearts 4 of Spades
K of Diamonds 8 of Clubs 8 of Hearts 5 of Diamonds J of Hearts
7 of Clubs 7 of Spades 7 of Hearts 3 of Spades T of Clubs
```

```
8 of Diamonds T of Diamonds J of Diamonds K of Hearts 9 of Hearts
3 of Clubs K of Spades A of Hearts 8 of Hearts 5 of Hearts
5 of Diamonds 9 of Hearts 6 of Hearts 8 of Hearts 4 of Spades
J of Clubs T of Diamonds 5 of Hearts 7 of Clubs A of Spades
8 of Hearts 7 of Diamonds J of Spades 7 of Spades 2 of Hearts
3 of Spades 3 of Diamonds 5 of Clubs 6 of Hearts 5 of Spades
8 of Clubs 9 of Spades J of Diamonds Q of Clubs J of Spades


-------------------- Card List ---------------------

4 of Clubs 4 of Clubs 4 of Hearts 4 of Hearts 6 of Clubs
6 of Clubs 7 of Diamonds 7 of Diamonds 8 of Diamonds 8 of Diamonds
T of Diamonds T of Diamonds T of Hearts T of Hearts Q of Clubs
Q of Clubs Q of Clubs Q of Clubs A of Spades A of Spades

-------------------- Cards After Removed ---------------------

T of Diamonds T of Diamonds T of Hearts T of Hearts Q of Clubs
Q of Clubs Q of Clubs Q of Clubs A of Spades A of Spades

--------------------- All Of One Card Removed --------------------

T of Hearts T of Hearts Q of Clubs Q of Clubs Q of Clubs
Q of Clubs A of Spades A of Spades

Press any key to continue . . .

--------------------------------------------------------------------------------
-*/
```

```
// CS 2B Lab 8
// Instructor Solution:
//  Original - Prof. Loceff, Updates, Edits, Annotations:&

// Notes:
//  - Use of sensible names for vars
//  - Faithfulness to spec
//  - Correct method qualifications (including virtuals)

#include <iostream>
#include <string>
#include <ctime>
#include <list>
using namespace std;

// class Card prototype ----------------------------------------
class Card {
public:
    enum Suit { clubs, diamonds, hearts, spades };
    static char DEFAULT_VAL;
    static Suit DEFAULT_SUIT;
    static const int NUM_CARD_VALS = 13;
    static const int NUM_CARD_SUITS = 4;

private:
    char value;
    Suit suit;
    bool errorFlag;

public:
    Card(char value = DEFAULT_VAL, Suit suit = DEFAULT_SUIT);

    bool set(char value = DEFAULT_VAL, Suit suit = DEFAULT_SUIT);

    char getVal() const { return value; }
    Suit getSuit() const { return suit; }
    bool getErrorFlag() const { return errorFlag; }
    bool equals(const Card& card) const;
    string toString() const;

    // helpers
private:
    static bool isValid(char value, Suit suit);

public:
    // comparison members and methods
    const static char valueRanks[NUM_CARD_VALS];
    const static Suit suitRanks[NUM_CARD_SUITS];

    int compareTo(const Card& other) const;

    static int getSuitRank(Suit st);
    static int getValueRank(char val);
```

```
};

// global scope methods ----------------------------------------
typedef list<Card> CardList;
void showList(const CardList& myList);
void insert(CardList& myList, const Card& x);
bool remove(CardList& myList, const Card& x);
bool removeAll(CardList& myList, const Card& x);

// for easy comparisons
int operator==(const Card& first, const Card& second) {
    return first.compareTo(second) == 0;
}

// for client Card generation
Card generateRandomCard();

char Card::DEFAULT_VAL = 'A';
Card::Suit Card::DEFAULT_SUIT = Card::spades;

// for comparisons -- ordering values and ranks
//  const forces correct # initializers
const char Card::valueRanks[NUM_CARD_VALS] = {
    '2', '3', '4', '5', '6', '7', '8', '9', 'T',
    'J', 'Q', 'K', 'A'
};

const Card::Suit Card::suitRanks[NUM_CARD_SUITS] = {
    Card::clubs, Card::diamonds,
    Card::hearts, Card::spades
};

Card::Card(char value, Suit suit) {
    // because mutator sets errorFlag, we don't have to test
    set(value, suit);
}

// stringizer
string Card::toString() const {
    string retVal = " ";  // just enough space for the value char

    if (errorFlag)
        return "** illegal **";

    // convert char to a string
    retVal[0] = value;

    if (suit == spades)
        retVal += " of Spades";
    else if (suit == hearts)
        retVal += " of Hearts";
    else if (suit == diamonds)
        retVal += " of Diamonds";
    else if (suit == clubs)
        retVal += " of Clubs";

    return retVal;
}

// mutator
bool Card::set(char value, Suit suit) {
    // convert to uppercase to simplify (may need to #include <cctype>)
    char upVal = toupper((int) value);

    if (!isValid(upVal, suit)) {
        errorFlag = true;
        return false;
    }

    // else implied
    errorFlag = false;
    this->value = upVal;
    this->suit = suit;
    return true;
}

// helper
bool Card::isValid(char value, Suit suit) {
    string legalVals = "23456789TJQKA";

    return legalVals.find(toupper((int) value)) != string::npos;
}

bool Card::equals(const Card& that) const {
    return this->value == that.value && this->suit == that.suit
        && this->errorFlag == that.errorFlag;

    return true;
}

// comparison method definitions ------------------------------------------
int Card::compareTo(const Card& that) const {
    if (this->value == that.value)
        return (getSuitRank(this->suit) - getSuitRank(that.suit) );

    return  getValueRank(this->value) - getValueRank(that.value) ;
}

int Card::getSuitRank(Suit st) {
    for (int k = 0; k < NUM_CARD_SUITS; k++)
        if (suitRanks[k] == st)
            return k;

    return 0;    // should not happen
}
```

```
int Card::getValueRank(char val) {
    for (int k = 0; k < NUM_CARD_VALS; k++)
        if (valueRanks[k] == val)
            return k;

    return 0;    // should not happen
}

// end of Card method definitions ------------------------------------

// global scope methods
Card generateRandomCard() {
    Card::Suit suit = (Card::Suit) ( rand() % Card::NUM_CARD_SUITS );
    char val = Card::valueRanks[ rand() % Card::NUM_CARD_VALS ];

    return Card(val, suit);
}

void insert(CardList& myList, const Card& x) {
    list<Card>::iterator iter;

    // loop until we find a float > x
    for (iter = myList.begin(); iter != myList.end(); iter++)
        if (x.compareTo(*iter) < 0)
            break;   // found the exact place for this card

    myList.insert(iter, x);
}

bool remove(CardList& myList, const Card& x) {
    list<Card>::iterator iter;

    // loop until we find or exhaust list
    for (iter = myList.begin(); iter != myList.end(); iter++)
        if (x == *iter) {
            myList.erase(iter);
            return true;
        }
    return false;
}

bool removeAll(CardList& myList, const Card& x) {
    list<Card>::iterator iter;

    // loop until we find it or exhaust list
    // reason we use iterator is so return bool gives information
    for (iter = myList.begin(); iter != myList.end(); iter++)
        if (x.compareTo(*iter) == 0 ) {
            //  removes ALL occurences. if only wanted first, use erase(iter)
            myList.remove(x);
            return true;
        }
    return false;
```

```
}

void showList(const CardList &myList) {
    list<Card>::const_iterator iter;

    cout << endl << " ----- Here's the List -----" << endl;
    for(iter = myList.begin(); iter != myList.end(); iter++)
        cout << "[" << (*iter).toString() << "] ";

    cout << endl << " ----- That's all! ----- " << endl << endl;
}

// ---- Test Driver -----

int main()
{
    CardList myList;
    Card cardSaveArray[10];

    // build list of 10 random Cards x 2 duplicates of each
    for (int k = 0; k < 10; k++) {
        cardSaveArray[k] = generateRandomCard();
        insert(myList, cardSaveArray[k]);
        insert(myList, cardSaveArray[k]);  // force duplicates
    }

    // should be sorted
    showList(myList);

    // remove 3 cards
    for (int k = 0; k < 3; k++) {
        // will be duplicates from forced doubles plus possibly rand gen
        while (remove(myList, cardSaveArray[k]))
            cout << cardSaveArray[k].toString() << " removed\n";

    }
    showList(myList);

    if (!removeAll(myList, cardSaveArray[0]))  // should have no effect
        cout << cardSaveArray[0].toString() << " not in list as expected. "
        << endl;
    else
        cout << " *** ERROR:  " << cardSaveArray[0].toString()
        << " was found, but shouldn't have been. "
        << endl;

    if (removeAll(myList, cardSaveArray[9]))  // (usually) will remove a card
        cout << cardSaveArray[9].toString() << " was  in list as expected. "
        << endl;
    else
        cout << " *** ERROR:  " << cardSaveArray[9].toString()
        << " was not found but should have been. "
        << endl;
    showList(myList);

    return 0;
}

/* --------------------- run Card compareTo() test -----------------

 ----- Here's the List -----
 [2 of Spades] [2 of Spades] [5 of Clubs] [5 of Clubs] [5 of Clubs] [5 of Clubs] [6 of
Spades] [6 of Spades] [7 of Clubs] [7 of Clubs] [8 of Spades] [8 of Spades] [J of Spades] [J
of Spades] [Q of Clubs] [Q of Clubs] [Q of Diamonds] [Q of Diamonds] [K of Hearts] [K of
Hearts]
 ----- That's all! -----

 6 of Spades removed
 6 of Spades removed
 Q of Diamonds removed
 Q of Diamonds removed
 K of Hearts removed
 K of Hearts removed

 ----- Here's the List -----
 [2 of Spades] [2 of Spades] [5 of Clubs] [5 of Clubs] [5 of Clubs] [5 of Clubs] [7 of
Clubs] [7 of Clubs] [8 of Spades] [8 of Spades] [J of Spades] [J of Spades] [Q of Clubs] [Q
of Clubs]
 ----- That's all! -----

 6 of Spades not in list as expected.
 7 of Clubs was  in list as expected.

 ----- Here's the List -----
 [2 of Spades] [2 of Spades] [5 of Clubs] [5 of Clubs] [5 of Clubs] [5 of Clubs] [8 of
Spades] [8 of Spades] [J of Spades] [J of Spades] [Q of Clubs] [Q of Clubs]
 ----- That's all! -----

 Program ended with exit code: 0

 --------------------------------- */
```