**A game of Bridge**

Bridge uses a pack of 52 playing cards. There are 4 suits (spades, hearts, diamonds, clubs) each of 13 cards: 1 (the Ace) to 10 and Jack, Queen, King. The Ace is the highest card, followed by the King, Queen, down to the 2. In bridge there are 2 teams of 2 players.

[[hide step-by-step guide]](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/bridgeexercise.php#Cards)

* Create a class called Card to represent a card. It will need 2 fields: one to store the card's value (1-13) and one to store the suit.

Think about what type of variables these fields should be (the suit could be a string or some kind of integer). Make the fields public like the example.

* Create a vector of 52 cards. Call it pack. Set the value of the fields of these cards so that the vector represents a full pack. You can do this by using for loops to set the values of each of the 52 cards.
* Shuffle the cards. C++ has a routine to do this. It's called random\_shuffle and it's used in a very similar way to the reverse function in the example above. This is one of the bonuses of using the standard algorithms - once you know how to one of them, the others are used in much the same fashion.
* In bridge each player's set of cards is called a *hand*. The following code creates a vector of 4 hands (hand[0] ... hand[3]), each of which is a vector of cards.
* const int numberOfPlayers=4;
* vector< vector<Card> > hand(numberOfPlayers);

These vectors are initially empty. Write some code that deals the cards out from the shuffled pack so that each player has a hand of 13 cards. You could do this using a for loop to add each card in turn to a hand. Note that if you have a card called c and you want to add it to the first hand you can use

hand[0].push\_back(c);

It's worth checking that at the end of the deal each hand has 13 cards - hand[0].size() will tell you how many cards are in the first hand.

* In bridge people often use a evaluation system where an Ace is worth 4 points, a King 3, a Queen 2 and a Jack 1. Write a function that given a hand of cards evaluates it, returning the number of points it's worth. Compile and run it. Think of a good name for the function. Decide how many input parameters it needs, and what type of variable it's going to return. To check whether it's working you could see whether the sum of all the hands' points is 40.
* In bridge the teams try to win *tricks* (a set of 4 cards, one put down by each player sequentially). Players should try to "follow suit" - e.g. if the first card of a trick is a Spade, subsequent players must put down a Spade if they have one. In our game the winner of the trick is the person who puts down the highest card with the suit of the first card (like "No Trumps" in bridge). We'll use the following strategy initially
  + The first player puts down a random card
  + Subsequent players will try to follow suit. If they can't, they put down a random card. If they can follow suit then they put down the lowest card they have if they can't beat the currently best card, otherwise they put down their best card.

A function to play a card will need to know what cards are available to play, the first card put down, and the best card put down so far. The card it puts down will have to be removed from the hand, so the hand parameter needs to be passed by reference (allowing it to be changed). If the card put down is the best so far, then that variable will need changing too. Write a function to implement this with the following prototype

bool playACard(vector<Card> &hand, Card firstCard, Card &bestCardSoFar);

Make it return true if the card put down is the best card so far. This requires about 25 lines of code and is best tackled in stages. First find out if the hand has any cards the same suit as the first card. Then find out whether the best card in that suit is better than the best card so far played. Test the program by creating cards with particular values to pass as the first card and the best card so far. To remove a card from a hand you can use something like

hand.erase (hand.begin()+n);

where n is the index of the card in the vector.

* The following function plays tricks using the code you've already written. Use it to test whether your code works.
* void play(vector< vector <Card> > hand) {
* int leader =0; // who plays the first card of a trick
* int trickwinner=leader;
* for (int trick=0;trick<13;trick++) {
* // pick a random card from the leader's hand
* int any=random() % hand[leader].size();
* Card firstCard= hand[leader][any];
* // remove that card from the player's hand
* hand[leader].erase(hand.begin()+any);
* Card best=firstCard; // the best card so far is the first card
* // now make the other 3 players put a card down
* for (int i=1;i<numberOfPlayers;i++) {
* // work out who the next player is
* int nexttoplay=(leader+i) %numberOfPlayers;
* if(playACard(hand[nexttoplay], firstCard, best) ) {
* // if this card wins the trick, record who won
* trickwinner=nexttoplay;
* }
* }
* leader=trickwinner;
* cout << "Player " << trickwinner << " won trick " << trick << endl;
* }
* }
* Add some code to print out how many tricks each team wins (players 0 and 2 are one team, players 1 and 3 the other). Check that a total of 13 tricks are won.
* Try to improve the first player's strategy by making them play the best card they have rather than a random card.
* Now let's introduce the idea of "trumps". In bridge a suit is often chosen to be the trump suit. Create a variable called trumpsuit and set it at random to one of the suits. Write a function called countTrumps that given a hand and the trump suit returns how many trumps are in the hand. If you use count\_if you can do this without using a loop.
* If a player can't follow suit they are allowed to "trump" the card by putting down a card that's a trump. If subsequent players also can't follow suit they are allowed to "overtrump" - put a higher trump card down. Change the playACard code to take advantage of the new possibilities.